ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

CAMPUS DI CESENA
SCUOLA DI INGEGNERIA E ARCHITETTURA

Corso di Laurea in Ingegneria Informatica

INTEGRATION PLATFORMS-AS-A-SERVICE PER INTERNET OF THINGS: IFTTT COME CASO DI STUDIO

Elaborata nel corso di: Sistemi Operativi

Tesi di Laurea di:
ALEX GAVATTA

Relatore:
Prof. ALESSANDRO RICCI

ANNO ACCADEMICO 2015–2016
SESSIONE II

PAROLE CHIAVE

IPaaS

Internet of Things

IFTTT

Ad Adele che mi ha spinto ad arrivare fino a qui, a Evelina e Agostino che ormai non ci speravano più, agli amici che ormai si erano dimenticati che io fossi iscritto

Indice

Introduzione

L'obiettivo nel futuro di Internet è quello di provvedere ad una infrastruttura per avere un immediato accesso alle informazioni sul mondo fisico e ad i suoi oggetti. Gli oggetti fisici possono essere applicati in molti ambiti diversi come ad esempio la telemedicina, gestione dei processi industriali, organizzazioni di ambienti e città etc. Ogni ambito di applicazione può contenere diversi tipi di dispositivi fisici ed ognuno di essi ha le proprie specifiche, che sono necessarie per potervi interagire. Per poter quindi riuscire in questo obiettivo è richiesta una visione a livelli che faciliti l'accesso ai dati prodotti da questa nuova tecnologia. IoT (Internet of Thing) è una visione che mira ad integrare le informazioni del mondo virtuale con quelle del mondo reale attraverso una struttura a livelli. Come le stesso termine IoT indica, la connessione ad Internet è una condizione indispensabile per utilizzare questo nuovo paradigma, che permette agli oggetti di divenire oggetti intelligenti connettendosi alla rete. Spesso questi oggetti hanno solo una connessione alla rete, spesso con specifiche molto limitate, ma non hanno tutto quello che serve per gestire la propria integrazione con il resto dell'ecosistema, siano esso servizi o software . Proprio per questo una soluzione di integrazione cloud, quindi già residente su Internet, il luogo dove gli oggetti diventano intelligenti, è l'ideale per abilitare IoT ad un utilizzo pervasivo e integrato. Queste soluzioni sono chiamate iPaaS, integration Platforms as a Service. Sfruttando metodi di wrapping che possano esporre API od utilizzando API native, gli iPaaS forniscono la parte mancante dello IoT, permettendo così l'utilizzo dei dati provenienti dagli Oggetti in maniera coerente con un workflow che si integra con tutti gli altri software e servizi utilizzati.
If This Than That (IFTTT) è un iPaaS, scelto per questa tesi come fulcro dell'analisi di questo nuovo paradigma, che permette una integrazione di vari servizi web e, soprattutto, oggetti di IoT.
L'obiettivo della tesi è quindi analizzare le necessità e le funzionalità di un

iPaaS che deve integrare vari endpoint IoT e, tramite l'utilizzo concreto di un iPaaS, mostrare in maniera pratica la metodologia per rendere un oggetto qualunque parte integrante di IoT, prima da oggetto a oggetto intelligente grazie ad Arduino e IFTTT e poi da oggetto intelligente a parte del paradigma IoT sempre grazie ad IFTTT.

Nel capitolo 1 verrà analizzato l'ambito IoT, i campi di utilizzo, l'architettura e le tecnologie abilitanti che ne permettono l'utilizzo.
Nel capitolo 2 invece verrà presa in considerazione uno dei più grandi scogli di IoT, l'integrazione. Per questo verrà discusso il cloud computing e l'iPaaS, la soluzione di integrazione residente nel cloud.
Nel capitolo 3 verrà analizzato e replicato il funzionamento di IFTTT, un semplice iPaaS, che pur non avendo tutte le grandi features dei più grandi servizi di integrazione, applica egregiamente il concetto di utilizzo di API, parte fondante dell'integrazione dello IoT.
Nel capitolo 4 verrà mostrato, sempre tramite IFTTT, utilizzando MKR1000, come rendere "intelligente" ogni singolo oggetto, e quindi poterlo rendere parte del paradigma IoT.

Capitolo 1

Internet of Things

1.1 Definizione

Un numero in continua crescita di dispositivi sono connessi ogni giorno ad Internet e nella visione di IoT esisteranno sempre più oggetti connessi in un modo o in un altro alla rete. La stessa Cisco mette come data di inizio dell'esistenza di IoT tra il 2008 e il 2009, cioè " il momento in cui a Internet hanno iniziato a essere connesse più "cose (o oggetti)" che persone" [14]. Questi oggetti sono quelli che quotidianamente usiamo e che ci circondano. Il passaggio da oggetti comuni a oggetti "smart" avverrà tramite tecnologia embedded, protocolli di comunicazione m2m, reti di sensori e attuatori, RFID etc.. e permetterà loro di integrarsi in maniera totalmente trasparente all'infrastruttura di rete, permettendo così di miscelare tra loro mondo fisico e mondo virtuale [3]. Saranno pervasivi in quanto onnipresenti in qualunque ambito ma totalmente invisibili e saranno in grado di raccogliere informazioni e interagire tra loro, cooperando per raggiungere uno scopo comune[18, 20].

Volendo utilizzare quello appena descritto come visione si può dire che il termine stesso IoT si può usare per definire:

1)una rete globale di oggetti "smart " che permetterà di estendere l'attuale rete Internet

2)una serie di tecnologie di supporto che permettano di realizzare questa visione (come detto prima, ad esempio, tecnologia embedded, protocolli di comunicazione m2m, reti di sensori e attuatori , RFID)

3) l'insieme di applicazioni e servizi che utilizzeranno queste nuove tecnologie [18, 2]

Ovviamente esistono moltissime altre definizioni su cosa sia lo IoT, ognuna data da un attore diverso in base all'ambito di interesse del proprio business o campo di ricerca. Eccone alcuni esempi:

- La definizione di EPoSS (the European Technology Platform on Smart Systems Integration): La rete formata da cose/oggetti che abbiano una identità, personalità virtuali che operano in spazi intelligenti utilizzando interfacce intelligenti per connettersi e comunicare con gli utenti, con i contesti sociali e con i contesti ambientali [13]

- La definizione di SAP: IoT ha l'obiettivo di creare un mondo dove gli oggetti fisici sono integrati in maniera trasparente nella rete delle informazioni, e dove gli oggetti fisici possono diventare partecipanti attivi del processo di business. I servizi sono messi a disposizione per interagire con questi oggetti intelligenti attraverso Internet, facendogli richieste oppure cambiando il loro stato ed ogni informazione ad essi associata, senza perdere di vista il fattore della sicurezza e della privacy associata a questo tipo di tecnologia [38]

- La definizione di CERP's (Cluster of European RFID Projects): Iot è una parte integrata del futuro Internet e può essere definita come una infrastruttura di rete globale dinamica, con capacità di auto configurazione basate su standard e con l'abilità di interoperare grazie a protocolli di comunicazione, dove oggetti fisici e virtuali hanno identità, attributi fisici e personalità virtuali ed utilizzano interfacce intelligenti e sono integrati in modo trasparente nella rete delle informazioni. In IoT, ci si aspetta che gli oggetti diventino partecipanti attivi nei processi di business, informatizzazione e sociali, dove avranno la possibilità di interagire e comunicare tra loro e con l'ambiente scambiando dati e informazioni apprese dall'ambiente stesso, mentre reagiranno autonomamente agli eventi del mondo reale/fisico, influenzandoli lanciando procedure che generanno azioni e servizi con o senza intervento umano. Interfacce, nella forma di servizi, faciliteranno le interazioni con questi oggetti intelligenti tramite Internet, permettendo di leggere e cambiare il loro stato, senza dimenticare fattori come sicurezza e privacy. [9]

- La definizione di CASAGRAS (Coordination and Support Action for Global RFID- related Activities and Standardization): IoT è una struttura di rete globale che lega oggetti fisici e virtuali attraverso lo sfrutttamento dei dati acquisiti e le capacità di comunicazione. Questa infrastruttura include gli sviluppi di rete e quindi di Internet presenti e futuri. Offrirà identificazione degli oggetti, sensori e connettività come base dello sviluppo di servizi e applicazioni che coopereranno. Questo sarà caratterizzato da un alto tasso di dati catturati autonomamente, trasferimento di eventi, connessione di rete e interoperabilità. [8]

Quindi, come si può dedurre, Internet delle cose sarà un progetto lungo e molto vasto, che ricoprirà sotto il proprio cappello un grande gruppo di campi di applicazione.

1.2 Campi di Applicazione

Il termine "Internet of Things" viene utililizzato per la prima volta nel 1999 da Kevin Ashton durante una presentazione che fece per Procter & Gamble, proponendo l'idea di legare l'uso di RFID con la catena di distribuzione della P&G. Dal '99 le tecnologie si sono molto evolute, facendo enormi progressi, e quindi anche la visione è cambiata da allora: da un metodo per tenere traccia di tutti gli spostamenti e posizionamento dei singoli oggetti, poco diverso dal concetto del codice a barre , a qualcosa di completamente diverso, estremamente pervasivo e ubiquo in molti campi. Come dice lo stesso Asthon dieci anni dopo: "[...]quello che intendevo, e ancora intendo, è questo: i computer odierni, e quindi Internet, sono totalmente dipendenti dagli esseri umani per le informazioni.[...]Se avessimo computer che sapessero tutto quello che c'è da sapere sugli oggetti, grazie a dati raccolti senza nessun ausilio da parte nostra, saremmo capaci di tracciare e contare tutto, riducendo in maniera drastica rifiuti, perdite e costi[...] Abbiamo bisogno di potenziare i mezzi di raccolta dei computer , in modo che possano vedere , sentire e odorare il modo per i fatti propri, in tutta la gloria della casualità[...] senza la limitazione dei dati inseriti dagli umani.". [4]

Quindi si può capire come per IoT si intenda qualcosa sviluppato e sviluppabile in una grande pletora di ambiti. Per questo sono tutte possibili speculazione di utilizzi futuri esempi come: RFID tag che dicono l'esatta ubucazione del proprio bagaglio, reti di sensori che monitorano il cambio

di concentrazione di sostanze tossiche o non tossiche nell'aria sistemi di riscaldamento, condizionamento e ventilazione che funzionano in maniera coordinata invece che in maniera indipendente, cerotti intelligenti che rilevano i microcambiamenti della condizione della pelle o del fllusso sanguigno, un dispositivo nell'autovettura che determina se il guidatore è troppo assonnato per guidare, sistemi di sorveglianza che analizzano quello che viene catturato e trova anomalie, occhiali per la realtà aumentata che forniscono informazioni aggiuntive su ciò che si sta guardando oppure descrivono quello che si sta guardando per aiutare gli ipovedenti, spazzolini che controllano che il livello di igiene orale sia soddisfacente.

Molte di queste cose sono già creabili ed utilizzabili, se non addirittura già create ed utilizzate. Ovviamente la visione di IoT non sta solo nelle tecnologie sopracitate in sè ma, come detto, nella creazione ed utilizzo autonoma e proattiva da parte dei singoli dispositivi dei dati generati dagli oggetti stessi, connettendoli a tutti gli effetti in un unico sistema, e permettendo agli oggetti stessi di agire autonomamente sul mondo circostante. Basti pensare solo ad alcuni esempi di questa profonda sinergia: l'autovettura sà già, grazie ad un dispositivo wearable, che ho dormito poco e automaticamente crea e mi invia tramite mail un percorso alternativo con i mezzi pubblici di cui avrà provveduto a comprare i biglietti o con un servizio di auto con guidatore chiamato automaticamente, il cerotto che rileva la pressione sanguigna in caso di problemi allerta automaticamente l'ambulanza inviando ai medici anche tutti i miei trascorsi clinici con patologie e allergie. Questa è una situzione ideale, a oggi futuristica, ma come visto quindi i campi di impiego saranno molteplici e profondamente interconnessi. Una macro divisione di mercato che tenga conto di questa profonda interconnessione è stata fatta dalla Machina Research:

- Intelligent Environment: edifici intelligenti / città smart e trasporti
- Intelligent Living: Automobili/ elettronica di consumo
- Intelligent Enterprise: Salute/ utilità/ manifattura/vendita al dettagio/ costruzioni/ agricoltura ed impianti estrattivi/ servizi emergenziali e sicurezza nazionale [1]

Purtroppo però allo stato attuale delle cose questa classificazione è troppo restrittiva, compattando insieme molte branche, il cui percorso di unificazione non sarà banale. Basti pensare a un progetto come la città smart,

dentro il quale finiscono innumerevoli campi di applicazione. Per definizione una città smart è una città che "utilizza tecnologie di informazione e comunicazione per rendere le strutture e i servizi critici dell'infrastruttura della città (amministrazione, educazione, sanità, sicurezza pubblica, edilizia, trasporti, servizi e così via) più consapevole, interattiva ed efficiente" [45] Alcuni esempi del modo in cui la tecnologia può influenzare differenti ambiti, necessari per mantenere una città "in buona salute":

- Sensori intelligenti che tengono traccia di oggetti e luoghi
- Applicazioni analitiche che smistano i dati ottenuti e permottono di usufruirne in maniera utile
- Tecnologie wireless e mobili per le intercomunicazioni
- Automazione di gestione di vari processi, da quelli emergenziale a quelli di normale routine

La IDC Government Insights, una società di analisi che aiuta governi e leader nel settore IT ad intraprendere le migliori scelte nel campo della tecnologia, ha pubblicato una lista delle competenze primarie necessarie per ottenere una città smart: pianificazione strategica della città, gestione della governance , interconnessione, mezzi e attrezzature, marketing , pubbliche relazioni e finanze. Non è difficile notare come la lista delle tecnologie necessarie definite per ottenere questi obiettivi sono quasi le stesse definite come necessarie per rendere possibile la visione di IoT (più avanti in questo capitolo). [21]

Una divisione che invece risulta essere più concreta e più attuale, forse meno visionaria ma decisamente legata alla divisione del mercato attuale, ci è fornita dalla Harbor Research. La classificazione data dalla Harbor Research divide IoT in 10 ambiti,descrivendone anche gli oggetti interessati:

- Edile: istituzionale/ commerciale/ industriale/ casalingo. Riscaldamento e condizionamento, anti incendio , sicurezza, ascensori, sistemi di controllo degli accessi, illuminazione
- Energetico : fornitura/ energie alternative/ ottimizzazione della richiesta. Turbine, generatori, stazioni di energia e commutatori

- Industriale: processo Industriale/ costruzione/ conversione/ assemblamento/ distribuzione/ processo di rifornimento. Pompe, valvole, cisterne, equipaggiamento per l'automatione e il controllo , tubature

- Sanità: Salute/ prevenzione/ ricerca. Apparecchiature elettromedicali, diagnostica, monitor, equipaggiamento chirurgico

- Rivendita: Negozi/ Accoglienza/ Servizi. Terminali nei punti vendita, distributori automatici , tag RFID, scanners e registratori di cassa,sistema di illuminazione e refrigerazione

- Sicurezza ed infrastruttura: sicurezza nazionale/ servizio nazionale/ difesa. Sistemi GPS , sistemi radar, sensori ambientali, veicoli, armamenti, schermatura dei confini

- Trasporti: veicoli da strada e fuori strada/ infrastruttura dei trasporti / logistica. Veicoli commerciali, aeroplani, treni, navi, segnaletica, pedaggi, tag RF, parchimetri, telecamere di sorveglianza, sistemi di tracciamento

- Informatica e networking: Aziende/ Data Centers. Switches, servers, storage

- Risorse primarie: agricoltura / sfruttamento minerario, petrolifero, gassoso e idrico. Tubazioni, equipaggiamento estrattivo ed agricolo

- Di consumo/ per professionisti: apparecchi/ elettrodomestici/ strumenti da ufficio/ elettronica di consumo. Dispositivi M2M , gadgets, smartphones, tablet PCs [37]

La crescita economica dei servizi basati su IoT dimostrano che ovviamente saranno uno dei campi futuri con una previsione di maggior crescita, con la sanità e il processo di produzione industriale a fare da volano. [28]

Ad esempio nel campo della sanità: si potrebbe avere, grazie alla tracciabilità , un miglioramento dei flussi dei pazienti all'interno degli ospedali , con smaltimento di code più rapido e maggior efficienza dei posti letto, un utilizzo migliore delle risorse , sia in termini di disponibilità che di posizionamento (avere attrezzature dove e quando servono) grazie alla tracciabilità delle attrezzature, senza incorrere nel pericolo di dimenticanza all'interno

del paziente. Se poi le stesse venissero anche identificate si potrebbero evitare furti o smarrimenti . Con l'acquisizione automatica dei dati si potrebbero evitare errori umani in fase di check in dei pazienti e sveltire ulteriormente il servizio, facendo check up precoci sulla disponibilità di attrezzature e medicinali che il paziente necessita per essere curato. Attraverso sensori applicati direttamente sui pazienti si potrebbe potenziare la telemedicina, offrendo un'assistenza del paziente a casa propria come se fosse virtualmente all'ospedale, controllare la reazione ai farmaci o avvisere in caso di mancata o errata somministrazione [44, 34].

Invece nel campo del processo industriale: tramite un utilizzo massivo di RFID si potrebbero identificare i pezzi in produzione. Utilizzando questi dati e associandoli a quelli del processo produttivo gli stessi macchinari saprebbero che ulteriori processi di trasformazione dovrà subire un pezzo per arrivare a completare gli ordini necessari presenti nel gestionale. Oltre a questo si potrà avere un controllo di anomalie durante il processo e l'eventuale blocco di tutto o parte della produzione, con l'aggiornamento automatico delle evasioni degli ordini, quindi migliorando tempistiche e sicurezza. [40]

Si potrebbero avere decine di divisioni possibili, ognuna portata a termine in base alla visione di coloro che hanno stilato l'elenco. Per trovare un terreno comune e cercare di sviluppare un processo congruo per la creazione del paradigma di IoT bisogna definire una o più visioni che possano definire in maniera esaustiva quello che IoT può rappresentare per tutti questi ambiti di sviluppo e queste diverse tecnologie finali.

1.3 Architettura di IoT

Gli oggetti smart possono essere considerati il blocco portante di IoT [24]. Molte definizioni di cosa è un oggetto smart sono state date:

- Gli oggetti smart sono artefatti di utilizzo quotidiano con in più l'abilità di elaborare e comunicare, potendo connettersi per scambiare informazioni su se stesso con altri artefatti e/o software [6]

- Gli oggetti smart dovrebbero essere in grado non solo di comunicare con altri oggetti smart e persone, ma anche scoprire dove si trovano,

quali altri oggetti sono in prossimità e che cosa gli è accaduto nel passato [30]

- Gli oggetti smart possono compiere azioni intelligenti e autonome basate su dati precedentemente collezionati ma possono anche aiutare le persone a prendere decisioni migliori e a compiere azioni più mature e responsabili [42]
- Gli oggetti smart sono identificabili univocamente, possono comunicare tra di loro, raccolgono e condividono dati [27]

Queste definizioni possono essere compattate ed espresse in maniera concisa tramite tre necessità essenziali che rendono un oggetto qualunque un oggetto smart:

- essere identificabili univocamente
- poter comunicare
- poter interagire (tra loro, con l'ambiente circostante, con altre entità della rete)

Quindi, ciò che differenzia gli oggetti smart da una qualunque entità ad oggi normalmente connessa a Internet, è la possibilità di poter interagire con l'ambiente circostante. Tramite sensori di svariati tipi si può avere un oggetto che sia conscio dell'ambiente che lo circonda, e tramite attuatori possa interagire direttamente con l'ambiente modificandolo in maniera attiva. Questa osservazione è fondamentale per capire alcuni delle necessità per lo sviluppo di IoT: l'inclusione di queste entità nella rete cambia completamente il concetto stesso di rete. Si avrà in questo modo una rete estremamente eterogenea, con l'inclusione di dispositivi che avranno una limitatà capacità di elaborazione e di comunicazione, in cui non tutto ciò che si trova al suo interno avrà tutti i layer del protocollo di rete e in cui le comunicazioni non saranno più solo end to end. Questo perchè in IoT si focalizzerà non più sulla comunicazione punto-punto ma su una rete di dati e informazioni, immesse ed utilizzate, da tutti gli apparati connessi, volgendo sempre di più verso una rete NDN (named data networking), cioè con architetture che usano i vari contenuti come pilastro fondamentale dell'architettura e dei principi [22].

IoT quindi si troverà a dover affrontare alcune criticità, mutuate dalle necessità degli oggetti smart:

- Capacità di localizzazione e tracciamento: ogni entità dovrà essere connessa tramite wireless a corto raggio, quindi si potrà tracciare la posizione e gli eventuali spostamenti nel mondo fisico. Un'altra tecnologia che permette questo prevede l'utilizzo di tag RFID

- Adattamento all'eterogeneità delle entità: come detto ci sarà nella rete un incremento delle diversità dei dispositivi collegati, sia a livello di elaborazione che di comunicazione

- Scalabilità: come visto dai numeri si avrà una esplosione di dispositivi connessi alla rete, tale per cui ci troveremo ad affrontare problemi come indirizzamento, protocolli, trasporto di dati e gestione dei processi concorrenti

- Bassi consumi: aumentando i dispositivi connessi aumenterà anche la necessità che ognuno di essi sia rifornito di energia elettrica, tramite batterie , e di un metodo di ricarica delle stesse, come micro pannelli solari. Ovviamente sarà un grande problema tanto che si dovrà cercare di ridurre al minimo i consumi in ogni ambito, a volte sacrificando anche parte della capacità di elaborazione o di comunicazione [25]

- Gestione dei dati: visto l'immensa mole di dati creati bisognerà fare in modo di trasformarli in informazioni utili, espresse in uno standard comprensibile ed accessibile al più grande numero possibile di dispositivi diversi utilizzando metadati per la descrizione dei contenuti, ottenendo così una classificazione sematica dei dati, utilizzabili autonomamente dai dispositivi che ne necessitano

- Sicurezza: con una pervasività così alta si avrà una dipendenza sempre maggiore dai dispositivi, e quindi anche il livello di sicurezza dovrà essere incrementato e soprattutto implementato fin dalle prime fasi del dispositivo, come valore fondamentale per poter essere utilizzato

- Autonomia: la complessità di alcuni contesti diventerà ingestibile per gli esseri umani quindi si dovrà avere la possibilità di avere delle reti di oggetti che si autogestiscono senza alcun intervento da parte dell'utente, se non in caso di guasti o di riprogrammazione [11]

Gli elementi fondanti quindi di IoT sono i seguenti:

- Identificazione: è cruciale poter referenziare l' oggetto in modo da poterlo trovare e richiamare a richiesta. Alcune tecnologie per l'identificazione possono essere l' uCode (ubiquitus code) e l' EPC (electronic product code) mentre per l'inidirizzamento ipv4, ipv6 (con la tecnologia di compressione 6LoWPAN)

- Rilevamento: l'oggetto deve aver la possibilità di raccogliere dati e interagire con l'ambiente in cui è inserito, inviando e ricevendo dati memorizzati in un database, in un data store o nel cloud. Alcuni esempi possono essere sensori e attuatori , sia normali che indossabili (per applicazioni wearable) che intelligenti (per applicazioni smart home)

- Comunicazione: Si deve chiaramente aver la possibilità di comunicare con l'oggetto e ricevere i messaggi che quest'ultimo ci invia. Essendo solitamente oggetti mobili, la maggior parte delle comunicazioni avviene attraverso tecnologie wireless come ad esempio WiFi, BlueTooth, IEEE 802.15.4, rete 4G , RFID e NFC.

- Elaborazione: le cpu e il software rappresentano il cuore dei nostri oggetti e gli permette di fare tutto quello per cui è stato progettato. Sul profilo hardware abbiamo alcune piattaforme come Arduino, Raspberry Pi, Intel Galileo e BeagleBone. Oltrettutto esistono molti software che vengono utilizzati per permettere funzionalità IoT. In questo campo abbaimo svariati SO, molto importanti in quanto in funzione per l'intero ciclo di vita del dispositivo , ed alcuni esempi sono Android , LiteOS and TinyOS. Ovviamente la potenza di calcolo può essere spostato in cloud, con il dispositivo che invia i dati che vengono elaborati in real time su cloud. Alcuni esempi sono AWS , Microsoft Cloud , Hadoop, Oracle.

- Servizi: I servizi di IoT possono essere categorizzati in 4 classi:

  Servizi relativi all'identificazione: permettono di portare un oggetto reale nel mondo virtuale identificandolo

  Servizi di aggregazione dei dati: collezionano e catalogano tutti i dati grezzi percepiti dai vari sensori (smart grid)

Servizi di collaborazione reattiva : utilizzano i servizi di aggregazione dei dati e permettono di ottenere dati e decidere ed agire conseguentemente (smart home)

Servizi onnipresenti: estendono il concetto di servizio di collaborazione reattiva rendendolo disponibile a chiunque, dovunque ed in qualunque momento [46, 17]

Alcuni dei servizi disponibili sono: BAS (sistemi di gestione e controllo di smart building), ITS (intelligent trasportation system che permette una gestione della rete di trasporti).

- Semantica: è la capacità di dare senso, permettendo di ottenere i servizi voluti, ai dati elaborati. E' una parte molto importante di IoT perchè permette ad una miriade di dati di acquisire il valore di informazione. Alcune tecnologie sono il RDF (Resource description Framework) e OWL (Web Ontology Language), ovviamente mutuate dal semantic web.

Per rappresentare come visione architetturale gli elementi fondanti di IoT, si può dividere questa architettura in svariati layers. Esistono due classificazioni, entrambe fondate su tre layers, che sembrano essere concise ma esaustive. Sono molto simili come layers ma affrontano la complessità dell'architettura di IoT da due differenti prospettive: una tende ad essere una visione più tecnica dell'architettura del sistema mentre l'altra considera come discriminante la catena del valore, quindi cosa viene aggiunto in ogni layer per aumentare il valore dell'oggetto in questione.

Il primo viene proposto come un insieme di visioni:

- orientata all'oggetto
- orientata a Internet
- orientata alla semantica

Il punto di vista "orientato all'oggetto"e quello "orientato a Internet" sono nati dagli specifici interessi e dalle finalità dei vari attori (che possono essere aziende, gruppi di ricerca, organismi di standardizzazione etc.). Il terzo punto di vista invece, quello "orientata alla semantica", è nato dalla complessità e dalla mole di dati che vengono dalla creazione dei due ambiti precedenti. I

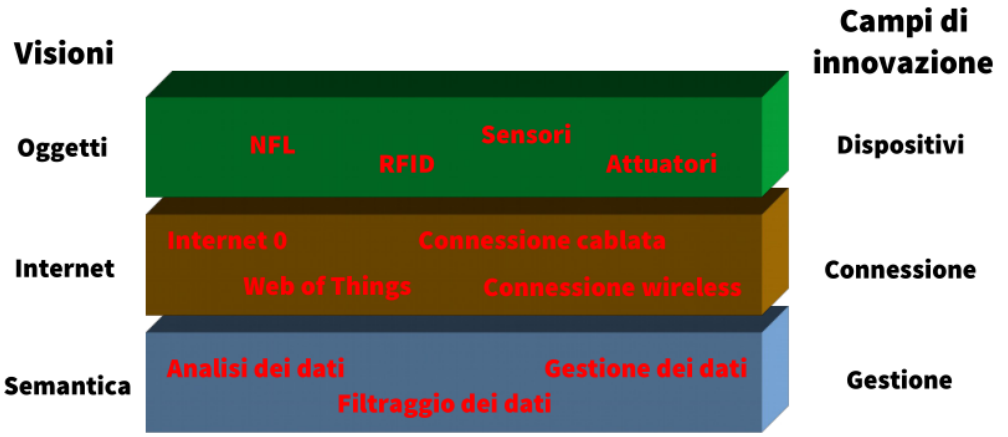

Figura 1.1: Comparazione tra diverse suddivisioni logiche dei layers di IoT

dati devono essere organizzati in modo da divenire informazioni usufruibili dagli oggetti che ne avranno bisogno [18].

Analizziamoli in maniera più approfondita: Parlando di "Internet of Thing" naturalmente al primo posto della definizione c'è il concetto di oggetto. L'oggetto è definito semplicemente come qualcosa di fisico che possiede un tag RFID. Questa definizione è chiaramente mutuata da Kevin Ashton, inventore del termine "Internet of Thing" e impiegato presso Auto-ID, laboratori di ricerca accademica sull'utilizzo degli RFID e di nuove tecnologie di rilevazione. L'attività principale del laboratorio è quella di sviluppare il più possibile la tecnologia RFID, utilizzata per fornire la tracciabilità, il monitoraggio dello stato dell'oggetto etc.., in modo che diventi uno standard per la produzione industriale utilizzata a livello mondiale [36]. Questo anche grazie all'integrazione che hanno avuto con EPC(Electronic product code), uno standard che permette , appunto utilizzando tag RFID, di identificare gli oggetti in maniera univoca in tutto il mondo. Questa visione "oggetto-centrica" non è sbagliata, ma è chiaramente solo parte di quello che IoT vuole essere. Infatti nella visione di IoT si vuole unificare il mondo fisico con quello virtuale e questo si può fare chiaramente solo utilizzando un'interconnessione tra i due mondi che si appoggi sulla rete esistente ma che si evolva per affrontare l'ingresso dei nuovi dispositivi al suo interno. Quindi nella visione "Internet-centrica" si analizza come l'oggetto si

interconnette al resto della rete per creare, come detto, un'unica rete ibrida (reale-virtuale). Come detto, le necessità di queste connessioni e quindi di questa rete saranno quelle di utilizzare, a fianco dei protocolli di rete già esistenti ma non sempre applicabili a tutti gli oggetti, protocolli wireless a basso consumo (come il 6LoWPAN o Internet 0) che, come proposto dalla IPSO (IP for smart object, un consorzio di compagnie che propone di utilizzare lo stack IP per interconnettere anche gli oggetti di IoT) , saranno una semplificazione ed alleggerimento del protocollo attuale. I dati che saranno generati da questa enorme mole di oggetti che verranno connessi e integrati sarà colossale. Con questo deriverà anche la necessità di organizzare, gestire , dividere i dati creati e memorizzati. Quindi nella visione "semantico-centrica" si penserà a come trasforamare ed usare il dato grezzo in informazione fruibile ed utile per analisi, decisioni ed azioni future, sia da parte dello stesso oggetto ma soprattutto da parte di altri oggetti che beneficeranno di questa conoscenza. [43]

Come detto esiste un'altra prospettiva che include tre layers:

- Dispositivi
- Connessione
- Gestione

In questa visione, che ad un primo sguardo può sembrare molto simile alla precedente, in realtà si analizzano i possibili campi di innovazione e sviluppo, a livello di valore aggiunto all'oggetto, per una attuazione di IoT [47]. Esistono grandi compagnie che sono presenti in ognuna delle tre categorie fornendo soluzioni per tutta la catena del valore di IoT, mentre altre forniscono valore aggiunto in solo una o due delle categorie sopracitate. Analizziamo più in dettaglio questa architettura. I dispositivi sono ovviamente gli oggetti fisici e possono essere divisi in due macro categorie: quelli che hanno intrinsecamente già una capacità computazionale e quelli che invece non hanno nessun tipo di capacità di elaborazione e che devono essere "abilitati" per diventare oggetti smart (come mobilia e animali). Nel secondo caso ad esempio si potranno utilizzare degli RFID tag per permettere questa abilitazione, inserendo così l'oggetto nella rete di IoT. Considerando invece il primo caso si potrà avere, inoltre, un utilizzo di sensori intelligenti(ad esempio sensori di temperatura, barometri , sensori di luminosità, di forza e pressione , etc.) e di attuatori per poter percepire e modificare l'ambiente.

Inoltre si potrà avere anche componentistica per il "power harvesting", ovvero riuscire a carpire energia elettrica da varie fonti(ad esempio RF, usati per la raccolta di energia tramite RFID tag, pannelli solari, carica ottenuta con vibrazioni e movimento etc.) Anche parlando di connessioni si possono fare due macro divisioni: rete wired e rete wireless. Con l'avanzare della tecnologia molto probabilmente la rete wired sarà relegata ai dispositivi non appartenenti a IoT ma a quelli appartenenti alla categoria dei dispositivi connessi (ad esempio computer, switch , router, firewall). Per ora però abbiamo ad esempio lo SCADA (Supervisory control and data acquisition), una rete di sensori collegati, che funzionano ancora principalmente con tecnologia wired. Invece le tecnologie che sono e verranno più utilizzate per connettere gli oggetti smart, soprattutto grazie alle piccole dimensioni che un trasmettitore wireless ormai può raggiungere, saranno probabilmente WSN (wireless sensor network), RFID e , conseguentemente, anche le comunicazioni M2M (machine to machine), il protocollo ipv6 e il 6LoWPAN (ipv6 over low power personal area network) , così come Internet 0. Ma la parte che probabilmente potrà creare la maggior parte del valore aggiunto è la parte di gestione. Qui risiederanno non solo le applicazioni ma anche tutto quello che riguarda la trasformazione di dati in informazioni , il dare "intelligenza" agli oggetti che non ne hanno la capacità intrinseca e al far comunicare (nel senso di scambiare informazioni) l'insieme di dispositivi eterogenei che verranno connessi). Stiamo parlando di tecnologie e metodologie come SOA (service oriented architecture), SaaS (Software as a service), cloud computing e, quindi , di middleware. Tramite queste i device collegati potranno aumentare in maniera esponenziale le proprie funzionalità e utilità a livello di business.

Per ottenere questo sviluppo si ha la necessità di avere o sviluppare alcune fondamentali tecnologie.

1.4 Tecnologie Abilitanti di IoT

Esistono chiaramente molte tecnologie che aiutano lo sviluppo verso la piena messa in opera di IoT. Di seguito verranno elencate solo le fondamentali in quanto pilastri fondamentali del concetto stesso di IoT. Esse sono divise in due categorie:

- le tecnologie utilizzate per identificazione e comunicazione

- le tecnologie utilizzate per elaborazione e interfacciamento di risorse eterogenee

Nel primo campo possiamo annoverare RFID e WSN.

RFID Nel dettaglio la tecnologia RFID (Radio Frequence Identification Tecnology)è stata la prima ad essere implementata ed a dare il via al concetto di IoT. RFID è un tipo di tecnologia di identificazione automatica senza bisogno di contatto che identifica gli oggetti dalle frequenze radio e accede ai dati interessati senza intervento umano. Ha una capacità di memorizzazione discreta sia leggibile che scrivibile, ha un buon potere di penetrazione della materia, ha tempi di lettura veloci, una durata notevole ed una grande adattabilità all'ambiente, potendo essere utilizzato praticamente con qualunque tipo di oggetto ed addirittura essere umani e animali [41]. Grazie alla grande velocità di lettura può identificare oggetti in movimento e anche leggere tag multipli. E' composto da tre componenti:

- Tag elettronico: è quello che contiene i dati ed ognuno ha un codice univoco che lo identifica. Normalmente è composta da un chip con una antenna. Può essere passivo, nel qual caso non possiede alcun tipo di batteria e viene alimentato direttamente da un lettore di tag nelle vicinanze, che tramite le onde elettromagnetiche trasmesse creano una corrente nell'antenna che trasmette indietro il proprio identificativo. Può essere semi-passivo nel qualcaso l'antenna è sempre alimentata dalla vicinanza del lettore stesso ma si ha la presenza di batterie che forniscono energia al chip. Altrimenti può essere attivo, nel qual caso le batterie alimentano sia il chip che l'antenna, propagando il segnale e rendendolo di più ampia portata [39].

- Lettore: dispositivo che legge e scrive i tag e, come detto, alimenta le antenne di alcuni tipi di tag.

- antenna: trasmette il segnale redio tra il tag e il lettore. Molto spesso sono associati all'uso del sistema ECP (Electronic Product Code)che è a tutti gli effetti tra il gestionale di produzione e i tag RFID [10].

WSN Per wireless sensor network si intende una rete di sensori che comunicano tra loro in maniera wireless. Ogni sensore può essere usato dagli

altri sensori come tramite per portare l'informazione a dei nodi speciali detti Sink, da qui poi verranno trasmessi ai server. Il vantaggio di questa tecnologia è l'utilizzo di sensori già intelligenti, quindi a differenza dei tag RFID hanno capacità computazionale, con il quale riescono già a elaborare parte dei dati ricevuti, scremando quelli inutili e inviando solo quelli richiesti. Anche il raggio di portata dei singoli nodi è decisamente maggiore, e questo è dovuto al fatto di essere alimentati. Purtroppo questo è anche un grande svantaggio: molto spesso i sensori sono molti e posti in posizioni difficili da raggiungere quindi un cambio di batteria sarebbe estremamente difficile. Per cui molto spesso i sensori sono per la maggior parte del tempo in standby, per cercare di prolungare il più possibile la propria vita. Oltettutto la tecnologia WSN non segue uno standard predefinito ma ogni volta che viene impegato deve venire ottimizzato per la realtà su cui è applicato [35].

Middleware Il termine middleware deriva dal campo del distributed computing e si riferisce ad un insieme di servizi come API, protocolli e servizi di infrastruttura che possano supportare lo sviluppo di servizi distribuiti. Esso è un livello che si posiziona tra il sistema operativo e l'applicazione e ne permette l'interoperabilità. Come lo definisce Gartner in maniera formale "un software di sistema che permette direttamente l'interazione a livello di applicazione tra programmi in un sistema distribuito" [15]. In maniera meno formale e più chiara lo si può definire come un layer di astrazione che permetta di superare l'etoregeneità, in modo che tramite il middleware gli utenti e le applicazioni possano accedere a dati e device di una serie di dispositivi interconnessi nascondendo tutti i problemi legati alla comunicazione e ad altri aspetti a basso livello dell'acquisizione dei dati stessi [7]. Quindi, riassumendo, i middleware applicati a IoT sono indispensabili per riuscire a superare le seguenti criticità:

- Difficoltà a definire e mantenere uno standard comune di tutti i diversi dispositivi che appartengono ai vari campi di IoT

- Le applicazioni di diversi domini necessitano di layer di astrazione / adattamento per poter essere utilizzate tra loro

- La mancanza di un interfacciamento, tramite API, tra il layer fisico degli oggetti e il servizio che si desidera ottenere dalle applicazioni, nascondendo tutto il processo nel mezzo.
- L'assenza di un collante per unire componenti diversi

Per ottenere i risultati esposti, un middleware deve soddisfare le seguenti caratteristiche:

- Gestione e scoperta di dispositivi: i dispositivi in gioco devono poter vedere la presenza degli altri intorno a loro, e viceversa, in modo tale da poter cooperare insieme
- Gestione dei dati: i dati raccolti sono molti e di svariate tipologie. Vanno quindi filtrati, gestiti e memorizzati in modo opportuno.
- Consapevolezza del contesto: il middleware deve essere consapevole del contesto in modo tale da fornire all'utilizzatore le informazioni necessarie nel momento in cui se ne ha bisogno. Questo obiettivo si può raggiungere in due passaggi:
  - Rilevazione del contesto: vengono raccolti dati e identificati fattori significativi che possono incidere nella risposta da fornire all'utente.
  - Elaborazione del contesto: dopo l'elaborazione viene scelta la risposta da fornire in base al contesto. Per implementare questi due passaggi si può procedere ad esempio in questo modo: ogni servizio supportato dal middleware si registra con un identificatore dipendente dal contesto. Questo identificatore viene cambiato in base ai cambiamenti di contesto (ad esempio quando si cambia locazione).
- Interoperabilità: in IoT si ha spesso a che fare con il bisogno di far condividere informazioni tra dispositivi (e applicazioni) eterogenei, che devono cooperare a uno scopo comune.
- Privacy e sicurezza: dato che in gioco c'è la raccolta di dati, questi devono essere protetti a dovere. Si possono realizzare reti peer to peer secure oppure implementare sistemi di autenticazione dei dati, gestione dei permessi e altro.

La parte che analizzeremo più da vicinò sarà la capacità del middleware di fornire l'interoperabilità tra dati e dispositivi, sfruttando la tecnologia cloud. Stiamo parlando dell'iPaaS, integration Platform as a Service, tecnologia emergente che analizzaremo più dettagliatamente nel prossimo capitolo.

Capitolo 2

Integration Platforms-as-a-Service

2.1 Introduzione

I dispositivi IoT (o gli endpoint IoT) non sono indipendenti, non sono soluzioni stand alone. Necessitano di essere gestiti, i dati che forniscono devono essere salvati, analizzati, distribuiti ed utilizzati in maniere consone, in più il flusso deve essere monitorato. Tutto questo può essere ottenuto implementando una piattaforma IoT tramite una suite di software o servizi cloud che facilitano le operazioni in cui si hanno degli endpoint IoT oppure delle risorse enterprise o cloud. Andando oltre, la maggior parte delle piattaforme IoT hanno già delle alcune caratteristiche embedded per l'integrazione, anche se spesso limitate. Queste caratteristiche possono essere abbastanza per un piccolo progetto ma non sono sicuramente sufficienti per più ampi progetti di integrazione IoT.
Infatti appena l'integrazione di IoT comincia ad aumentare in complessità e comprende integrazioni con applicazioni multiple e database, così come una gestione complessa delle API, è il momento di pensare ad una introduzione di un iPaaS, integration Platform as a Service.

2.2 Cloud Computing

Il cloud computing è un modello per avere un accesso da dovunque e su richiesta ad un pool condiviso di risorse computazionali (rete, server, disco, applicazioni e servizi) che possono esssere rapidamente forniti ed utilizzati con un minimo sforzo di gestione o interazione dal fornitore del servizio. Questo modello è composto da cinque caratteristiche essenziali:

- Self service on-demand. Un utente può unilateralmente decidere di incrementare e utilizzare maggiori risorse computazionali senza dover richiedere nessun tipo di intervento umano

- Accesso da dovunque. I servizi sono accessibili tramite la rete e devono essere accessibili tramite un meccanismo standard che permette l'accesso al più ampio numero di dispositivi possibili

- Raggruppamento di risorse. Le risorse che il fornitore del servizio offre sono raggruppate in modo che possano fornire diverse quantità di risorse computazionali a più utenti, venendo incontro alle richieste di ogni singolo utente.

- Rapida elasticità. Le capacità possono essere dinamicamente e tempestivamente variate e divenire istantaneamente utilizzabili, scalando verso l'alto o verso il basso a seconda delle esigenze puntuali delle richieste. Per l'utente le risorse devono sembrare infinite ed essere sempre eventualmente disponibili per l'utilizzo in caso di necessità.

- Servizi misurabili. Il sistema cloud automaticamente controlla ed ottimizza l'uso delle risorse tramite un sistema di monitoraggio che permette di sapere, tramite opportune astrazioni, le quantitè utilizzate delle stesse. Questo permette trasparenza sia per il fornitore che per l'utilizzatore sulle reali quantità di risorse utilizzate.

2.2.1 Modelli di distribuzione

Un servizio cloud può essere suddiviso tenendo conto del tipo di modelli di distribuzione. Essi tendono a distinguere il metodo di accesso, il proprietario delle risorse interne al cloud. Si possono dividere in 4 categorie:

- Cloud pubblico
- Cloud privato
- Cloud ibrido
- Cloud di community

Cloud pubblico L'infrastruttura cloud è accessibile al pubblico o a un grande gruppo industriale ed è di proprietà di coloro che vendono i servizi cloud [31]. Nel cloud pubblico le risorse sono offerte come servizio, di solito tramite una connessione ad Internet ed utilizzano una formula di pay-per-use. Gli utenti possono graduare l'ordine di grandezza dell'utilizzo del cloud su richiesta e non c'è bisogno di comprare hardware per utilizzare il servizio. I provider di servizi cloud pubblici si prendono in carico la gestione dell'infrastruttura e delle risorse in modo da fornire le quantità richieste dagli utenti stessi [26]. Un servizio di cloud pubblico risiede in Internet ed è implementato in modo tale da essere fruibile tramite una connessione alla rete, tentando di nascondere il più possibile la sua natura remota cercando di fornire una esperienza simile, a livello di capacità e di servizi, a quella locale Gli utenti del cloud pubblico sono tipicamente utenti così detti residenziali, che si connettono a Internet tramite una rete fornita da un ISP [5].
I maggiori benefici del cloud pubblico sono la disponibilità continua dei dati e dei servizi, l'abbattimento dello spreco di risorse e un setup che non richiede investimenti di hardware o tempo.
Per contro i più gravi difetti sono che nel cloud pubblico non si ha alcuna visibilità su dove siano, come siano protetti e come siano utilizzati, i propri dati e che , in caso di problemi al servizio di cloud, sia i dati che i servizi risultano inutilizzabili.
Alcuni esempi di cloud pubblico sono: Amazon AWS, Google apps, microsoft Office 365.

Cloud privato L'infrastruttura cloud privata è utilizzata solamente per una organizzazione. Può essere gestita dall'organizzazione stessa oppure da terzi e può esistere sia in loco che in remoto[31]. All'infrastruttura cloud posso accedere solo i membri della organizzazione e/o terze parti a cui è stato fornito il permesso. Lo scopo non è fornire servizi cloud al pubblico

ma fornirli solamente ad una organizzazione, come ad esempio un'impresa che vuole rendere disponibili i propri dati ai propri punti vendita. Un cloud privato è mantenuto nel data center della compagnia che lo utilizza e fornisce i propri servizi solo agli utenti all'interno della compagnia e dei propri partners, proprio per questo fornisce un livello di sicurezza estremamente maggiore a quello dei cloud pubblici e permette di utilizzare alcune delle risorse già presenti ma non sfruttate del data center. Rendere queste risorse non utilizzate disponibili attraverso un'interfaccia cloud permette di utilizzare gli stessi tool come se si lavorasse in un cloud pubblico ma di beneficiare dei paradigmi di gestione del cloud come il self service on demand delle risorse e la gestione automatica delle risorse computazionali.
Il più grande vantaggio rispetto al cloud pubblico è ovviamente l'aumentato livello di privacy e sicurezza che offre. Essendo i dati le risorse più importanti di una organizzazione questo fa capire quanto questo vantaggio non sia da sottovalutare.
I difetti invece si possono riscontrare nel costo, ovviamente maggiore, dovuto all'acquisto dell'hardware, dello sviluppo del software e della manutenzione del tutto.

Cloud ibrido I cloud ibridi sono più complessi degli altri modelli di distribuzione perchè comprendono la composizione di due o più cloud(privati, pubblici o comunitari). Ogni membro rimane una entità unica, ma è legata alle altre attraverso tecnologia standard o proprietaria che permetta la portabilità di applicazioni e dati tra i cloud [23]. Un cloud ibrido è la composizione di almeno un cloud privato e un cloud pubblico e di solito viene offerto in una delle due maniere: un vendor ha un cloud privato e forma una partnership con un provider di cloud pubblico oppure un provider di cloud pubblico fa una partnership con un vendor che fornisce cloud privati. Nel cloud ibrido, una organizzazione fornisce e gestisce alcune risorse internamente ed alcune risorse esternamente.
Idealmente, l'approccio ibrido permette di avere i vantaggi del risparmio dei costi e della scalabilità che offre il cloud pubblic senza esporre i dati e le applicazioni sensibili di una organizzazione alle vulnerabilità di terze parti [48] [19].
Come lati negativi ovviamente si ha che, espandendo i propri confini dell'IT all'esterno dell'azienda, si apre un'area più vasta ad attacchi e si dà una

parte del controllo della propria infrastruttura cloud ad un service provider esterno.

Cloud comunitario Il cloud comunitario è in un qualche modo simile al cloud privato ma l'infrastruttura e le risorse sono esclusive di due o più organizzazioni che hanno in comune privacy, sicurezza e regole interne invece che di una singola. Il cloud comunitario aspira a combinare la fornitura di risorse distribuite, di controllo distribuito e di consumi distribuiti con il metodo di utilizzo tipico del cloud. Si cerca di rimpiazzare i vendor di servizi cloud, reindirizzando le strutture e le strutture sotto utilizzate per formare appunto un cloud comunitario, con nodi che potenzialmente ricoprono tutti i ruoli tipici, cioè i consumatori, i produttori e i coordinatori di servizi cloud [29] [12].
Ovviamente si ha come vantaggi principali il fatto di avere un costo inferiore ad un cloud privato potendo comunque utilizzare una tipologia di servizio molto simile.
Per contro però, oltre a costare comunque di più di un cloud pubblico, si va incontro ad una divisione di risorse predefiniti tra più attori.

Cloud Storage Type	Host	Owner	Access	Users
Public cloud	service provider	service provider	Internet	public as individuals, organizations
Private cloud	Enterprise (Third Party)	Enterprise	Intranet, VPN	Business organizations
Hybrid cloud	Enterprise (Third Party)	Enterprise	Intranet, VPN	Business organizations
Community cloud	Community (Third party)	Community	Intranet, VPN	Community members

Figura 2.1: modelli di ditribuzione dell'infrastruttura dei servizi cloud

2.2.2 Infrastruttura

L'architettura cloud può essere divisa in 4 layer generali (Hardware, Infrastructure, Platform, Application), modellata seguendo l'architettura dei modelli service-oriented [31]:

- Infrastructure as a Service(IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

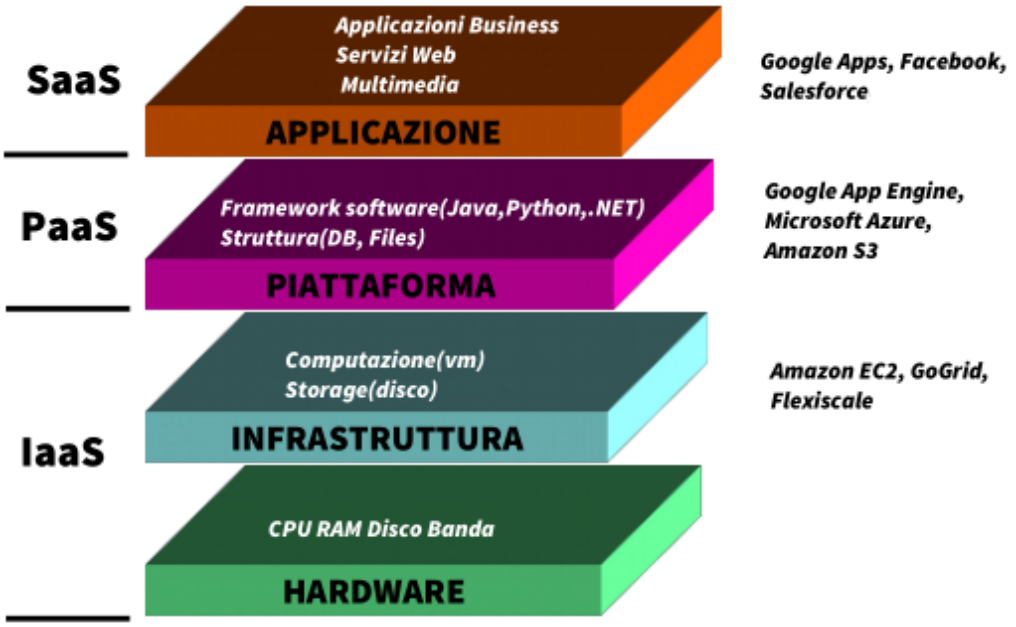

Figura 2.2: Layer dell'infrastruttura dei servizi cloud

Infrastucture as a Service E' il livello più vicino all'hardware e offre l'infrastruttura computazionale (come macchine virtuali e altri tipi di risorse) ai propri utenti. Questo permette agli utenti di astrarre dai problemi di gestione fisica (come ad esempio la sicurezza, i backup, lo spazio di memorizzazione e la potenza di calcolo, la connettività, il bilanciamento dei carichi etc.) e di pagare solo la quantità di risorse realmente utilizzate. Per sviluppare le proprie applicazioni gli utenti devono installare sia i sistemi operativi che i software necessari, così come è lasciato anche il mantenimento degli stessi (patch e aggiornamenti).
Alcuni esempi sono Amazon EC2, Enomaly, GoGrid, citrix Xen etc.

Platform as a Service Questo livello offre come base un ambiente di sviluppo per applicazioni già pronto, quindi non solo la parte di infrastruttira fisica è totalmente invisibile all'utente ma anche tutta la parte di microgestione di software e applicazioni (sistema operativo, database, web server, software di sviluppo etc.) che permettono la creazione della piattaforma richiesta. Eventualmente i software installati possono essere scelti dall'utente.
Alcuni esempi sono Microsoft Azure, Google app engine e Amazon S3

Software as a service Questo è il livello più alto di servizio offerto al cliente, in cui l'utente utilizza direttamente un software o un servizio senza gestire niente di tutta l'infrastruttura sottostante. Normalmente il software è installato e lanciato direttamente lato cloud, e di cui il vendor si prende in carico tutta la gestione dell'infrastruttura sottostante e della sua manutenzione e dell'ottimizzazione delle performance, dando all'utente un entry point, di solito anche esso remoto, unico che nasconda tutto questo.
Alcuni esempi sono Gmail, Facebook, Slack, Trello etc.

Un pezzo del puzzle che rimane escluso dall'architettura cloud è la funzione tipica del middleware, cioè l'integrazione tra le varie applicazioni e i vari dati. Ovviamente questa parte risulta indispensabile dal momento in cui si decide di spostare anche solo una parte del proprio ecosistema IT nel cloud. Infatti più le piattaforme si muovono sulla nuova architettura più aumenta il bisogno di interoparibilità tra tutti i vari componenti, siano esse applicazioni o dati. Bisogna infatti riuscire a garantire una integrazione non solo tra vari cloud ma anche tra cloud e le applicazioni on-premise su cui molto probabilmente è presente la maggior parte dei dati che dovranno essere utilizzati. Un middleware cloud di integrazione è la parte che soddisfa questo bisogno in questo tipo di infrastrutture. Ne esistono di due tipi:

- application Platform as a Service (aPaaS)
- integration Platform as a Service (iPaaS)

Il primo tipo, aPaaS, è una suite che è indirizzata a fornire agli utenti una piattaforma integrata di memorizzazzione e gestione di applicazioni individuali e di servizi forniti dalle applicazioni. Di solito offre un servizio cloud aggregato di server, strumenti di sviluppo, composizione, portale ed esperienza utente, organizzazione, gestione dei dati sicurezza delle applicazioni,

tutto in una unica entità. Del secondo tipo invece, cioè l'iPaaS, ovvero l'integrazione di applicazioni, ne parleremo più estensivamente nei prossimi capitoli.

2.3 Integration Platforms-as-a-Service

Secondo la definizione di Gartner, l'iPaaS è un modello emergente descritto come "Una suite di servizi cloud che permettono lo sviluppo, l'esecuzione ed il mantenimento del flusso di integrazione connettendo qualunque combinazione di processi, servizi, applicazioni e dati sia locali che basati su cloud" [16].
Per MuleSoft, uno dei rivenditori leader di middleware sul mercato descrive l'iPaaS come: "Una piattaforma per costruire e rilasciare integrazioni all'interno del cloud e tra il cloud e l'impresa. Con iPaaS, gli utilizzatori possono sviluppare un percorso di integrazione che connette applicazioni locali e remote e rilasciarle senza installare o gestire nessun hardware" [32].
Secondo MuleSoft le capacità indispensabili di un iPaaS sono le seguenti:

- 1. Intermediazione
  Il passaggio al cloud allarga chiaramente gli ambiti in cui sono necessari l'intermediazione dell'iPaaS, come per esempio applicazioni SaaS, servizi cloud, applicazioni custom sviluppate nel cloud stesso, senza dimenticare ciò che già prima esisteva, come le applicazioni, i servizi e le risorse on-premise. L'integrazione di applicazioni e servizi è il paradigma fondamentale di ogni iPaaS e quindi è il suo compito principale far comunicare tutti gli ambiti sopracitati tra loro.

- 2. Orchestrazione
  Per orchestrazione si intende il riarrangiare i vari componenti per ottenere il risultato desiderato. Probabilmente è proprio il caso d'uso principale dell'iPaaS, in cui i componenti sono i servizi SaaS, le applicazioni cloud e quelle on premise, che richiedono connettività e l'abilità di mappare i dati tra loro, permettendo così di ottenere un workflow che possa utilizzare indifferentemente tutti i dati, ovunque essi siano.

- 3. Contenitore dei servizi
  Oltre a integrare le applicazioni e i servizi tra loro gli utenti dell'iPaaS hanno bisogno di pubblicare i loro stessi servizi utilizzando delle API

come interfaccia. Utilizzando molte applicazione che pubblicano le proprie interfacce si aiuta a supportare differenti tipi di consumatori di servizi, come ad esempio browser, device mobili e comunicazioni M2M

- 4. Sicurezza
  E' un argomento vastissimo ed è uno dei maggiori punti di contenzioso nell'uso del cloud. La sicurezza di un iPaaS deve contenere svariati tipi di metodi , visto l'eterogeneità di quello che unisce. Alcuni esempi sono l'autenticazione e l'autorizzazione all'accesso delle risorse della piattaforma stessa, la gestione degli accessi alle applicazioni SaaS e cloud , criptare e memorizzare dati sensibili se usato in regime di multi-tenant, assicurare la sicurezza dei servizi pubblicati usando ad esmpio OAuth, SAML e WS-sec, supportare l'SSL, regole di gestione del firewall ed eventualmente l'accesso tramite VPN.

- 5. Data gateway
  Essendoci dati di decine di anni all'interno della LAN a cui si deve avere accesso tramite il servizi e applicazioni SaaS e cloud, l'iPaaS deve offrire un canale sicuro per l'accesso ai dati on-premise delle organizzazioni. Si deve quindi installare un gateway che fungerà da proxy per tutte le interazioni cloud/on-premise che richiedono accesso a risorse come database, file system , applicazioni o altri servizi

- 6. Ambiente di sviluppo
  In questa area tutto dovrebbe rimanere come è sempre stato, mantenendo l'ambiente, il processo e il metodo di sviluppo inalterato. Sia che sia utilizzato localmente o che sia rilasciato nell'iPaaS, il metodo utilizzato non dovrebbe cambiare, idealmente essendo indistiguibile l'uno dall'altro.

- 7. Monitor a runtime
  Questa parte principalmente è solo una cosa che riguarda le SLA(Service Agreement Level)tra fornitore e fruitore del servizio. Essendo un servizio fornito da terzi si deve avere modo di quantificare le risorse date a disposizione ed utilizzate da ciascun cliente. Queste informazioni possono essere, ad esempio, tempi di risposta prevedibili, limite di utenti concorrenti, limiti di traffico o utilizzo di banda etc. Si devono

avere allarmi per quando si è raggiunto una certa vicinianza al limite. Ovviamente questo tipo di monitor è molto diverso da quello standard, in quanto il fruitore del servizio non deve gestire le piattaforme, ma solamente sapere e capire le quantità utilizzate delle varie risorse.

Il modello architetturale di un iPaaS può essere suddiviso in 3 componenti principali:

- esterno: che include tutto quello che mette in rapporto diretto l'organizzazione con esterni. Ricopre ad esempio i servizi di ecommerce, il trading con i partner (B2B), Internet of Thing (IoT), dispositivi mobili etc. E' la fascia che fa parte dei system of engagement.

- integrazione: che include le funzioni espletate dai vecchi sistemi di integrazione locali, spostandoli nel cloud. Include principalmente Enterprise Service Bus (ESB) e Business Process Management (BPM) E' la fascia che fa parte dei system of interaction.

- applicazioni interne: che include tutto quello che riguarda i processi interni, sia decisionali che di analisi. Quello che viene incluso è la gestione di ciclo di vita del prodotto, digital operation, Customer Relationship Management (CRM), Enterprise Resource Planning (ERP) e data warehouse. E' ciò che viene definito come system of records.

La divisione/visione appena esposta mira ad essere una visione olistica dell'implementazione del modello iPaaS,valida per ogni tipo di cloud(pubblico privato o ibrido), ma non è detto che un iPaaS gestisca tutti questi ambiti e che, di conseguenza, possieda tutti e 3 i componenti principali. Generalmente le componenti può diffusamente coperte da un iPaaS sono quelle esterne e quelle di integrazione.

Gli ambiti di integrazione possono essere divisi in 3 macro categorie:

- Cloud to cloud: è l'integrazione nativa dell'iPaaS, che permette di integrare ovviamente due cloud

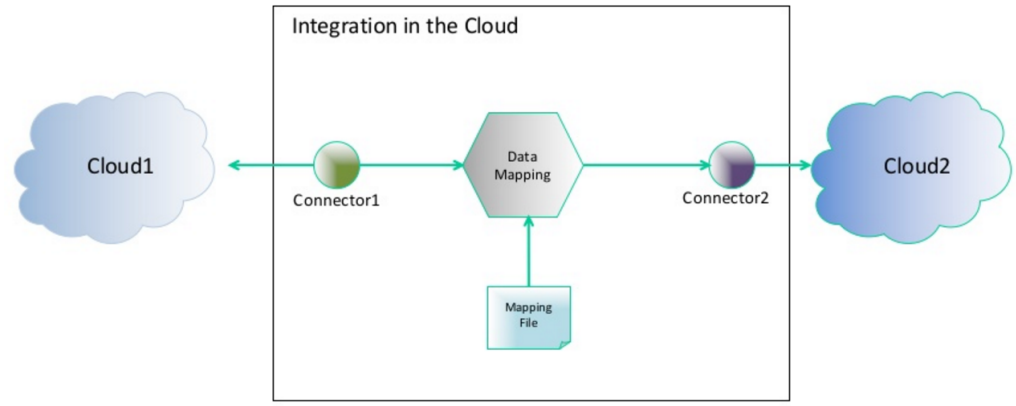

Figura 2.3: Integrazione cloud to cloud

- Cloud to on-premise o on-premise to cloud: quando una parte delle informazioni vitali di un'azienda vengono spostate su cloud questa è l'integrazione utilizzata e riguarda la comunicazione tra software e servizi legacy, locali e on-premise dell'organizzazione con il cloud stesso

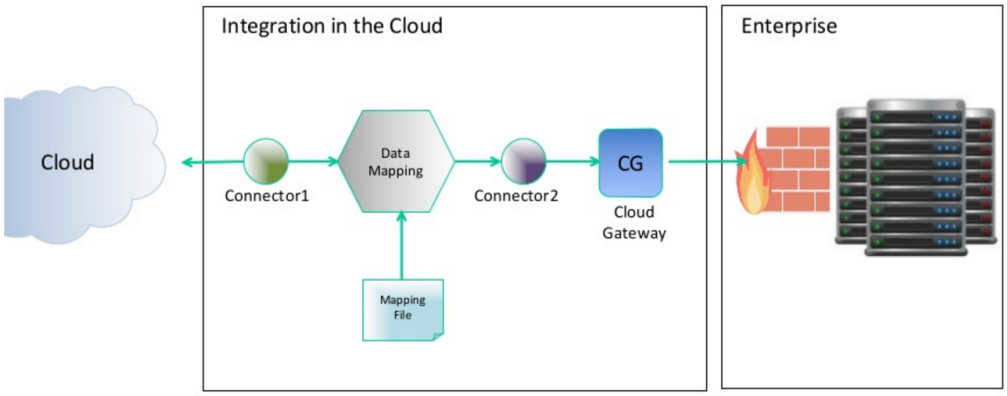

Figura 2.4: Intergazione cloud to on-premise

- On-premise to on-premise: è un ambito ancora poco sviluppato ma talvolta l'iPaaS fornisce una soluzione anche per integrare tra loro servizi e dati di applicazioni on-premise. Di solito per fornire questa funzionalità il "core" dell'integrazione, non dovendo mai uscire dall'interno dell'organizzazione, viene rilasciato anch'esso on-premise.

Per parlare di integrazione si deve sicuramente parlare di un concetto espresso con molti termini ma riassumibile con il nome generico di "connettore". Per connettore si intende quella parte di software che si prende in carico l'esporre (leggere e scrivere) i dati in una maniera standard e facilmente utilizzabile di un contenitore, ovunque esso si trovi. Generalmente, questo viene fatto in maniera standard, creando delle API che possono essere invocate per ottenere i dati o i servizi necessari.
Le API solitamente sono web based e principalmente possono essere di due tipi:

- REST. è un web service conforme ai paradigmi RESTful(vedi capitolo 3) ed è di tipo ROA(Resource Oriented Architecture). La chiamata delle api avviene tramite l'invocazione di un url in cui viene referenziato un singolo elemento(utente, account o dato) tramite il protocollo http e con un metodo ben preciso (GET, POST, PUT, DELETE)viene eseguita un'azione sullo stesso.

- SOAP. è un web service di tipo SOA(Service Oriented Architecture) ed espone una serie di metodi richiamabili da remoto da parte di un client. Utilizza il protocollo http semplicemente come protocollo di trasporto. E' derivato dalle tecnologie di interoperabilità esistenti al di fuori del Web e basato essenzialmente su chiamate di procedura remota, come DCOM, CORBA e RMI. In sostanza questo approccio può essere visto come una sorta di adattamento di queste tecnologie al Web [33].

Molte applicazioni SaaS possiedono già API che possano essere utilizzate per interfacciarsi in maniera automatica con esse, in questo caso l'iPaaS utilizza le API native, rendendo automatico e semplice, l'utilizzo del servizio o del dato. Invece molte applicazioni e servizi on-premise non possiedono interfacce standard ma solo proprietarie, accessibili spesso solo con software proprietari. Per questo come soluzione alcuni iPaaS aggiungono un layer di

wrapping che si prende in carico la parte legacy dell'applicazione o del servizio, esponendo delle API standard che possono essere utilizzate esattamente come quelle di un SaaS. Generalmente l'iPaaS tenta di rendere semplice e standard l'accesso a qualunque risorsa, dato e applicazione, in qualunque ambito, cloud o on-premise, in modo da poter facilmente creare un workflow che non trovi ostacoli di sorta nell'eterogeneità delle risorse trattate.

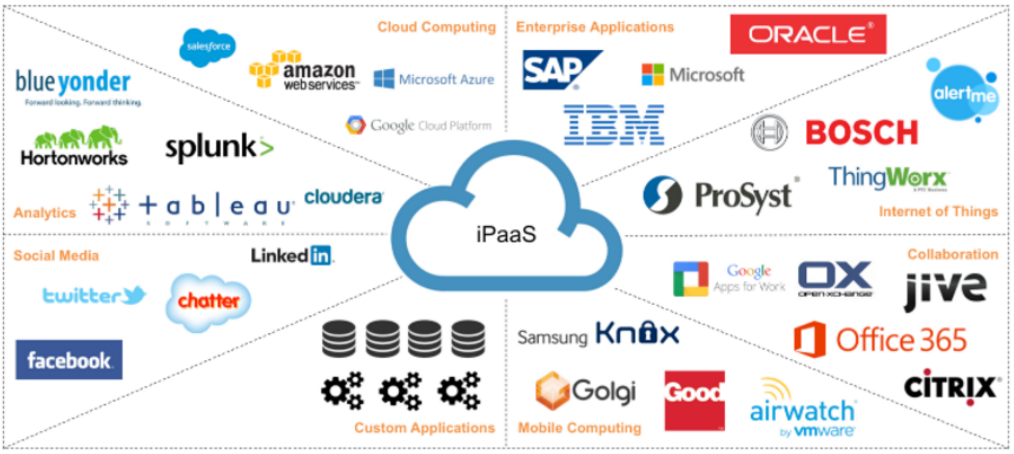

Figura 2.5: Possibili servizi connessi tramite iPaaS

2.4 IPaaS per IoT

L'utilizzo di cloud middleware da parte di IoT permette di superare molte delle sue intrinseche limitazioni [49]. Come possiamo notare dalla tabella le caratteristiche dei due sono in larga misura complementari.

2.4.1 Caratteristiche fondamentali

La tecnologia probabilmente non è ancora matura per creare una soluzione che permetta di superare tutte le varie complessità nell'ambito dell'implementazione e dell'integrazione per IoT, ma questo spesso è una cosa non necessaria. Le piattaforme IoT devono servire a dei ben precisi scopi, e coloro che li producono dovrebbero fare in modo che possano portarli a termine

	IoT	Cloud
Dislocamento	**Pervasivo**	**Centralizzato**
Raggiungibilità	**Limitata**	**Dovunque**
Componenti	**Oggetti nel mondo reale**	**Risorse virtuali**
Capacità computazionali	**Limitate**	**Virtualmente illimitate**
Disco	**Limitato o nessuno**	**Virtualmente illimitato**
Ruolo di internet	**Punto di convergenza**	**Utilizzato per fornire servizi**
Big Data	**Sorgente**	**Gestione**

Figura 2.6: Aspetti complementari del Cloud e di IoT

nel miglior modo possibile. Invece i fornitori di iPaaS devono avere tutt'altre competenze che, utilizzando una combinazione razionale di strumenti, permettano di integrare al meglio le tecnologie IoT al resto dell'ecosistema IT.
Come detto esistono molti tipi di iPaaS, alcuni specializzati in alcuni ambiti ma carenti in altri. Le caratteristiche essenziali che un iPaaS che vuole integrare anche endpoint IoT dovrebbe avere sono:

- Avere un approccio API-first nella tipologia di gestione dell'integrazione.
  Molte soluzioni IoT provvedono delle API per una migliore integrazione tra di loro ed altri sistemi o applicazioni, quindi ha senso che l'iPaaS utilizzi prima di tutto questa tecnica per connettere i dispositivi IoT. Purtoppo non tutte le problematiche di integrazione IoT possono essere risolte con un utilizzo di API, quindi bisognerebbe evitare un approccio API-only.

- Una architettura event-driven.
  Questo perchè molte piattaforme IoT devono avere, per poter essere utilizzate correttamente e proficuamente, un processamento e una risposta in tempo reale, cosa che una soluzione batch non può garantire

- Supportare sia integrazioni SaaS che integrazioni on-premise.
  Come ogni altra nuova tecnologia è impossibile sapere con certezza all'inizio di un progetto IoT quanta integrazione sarà necessaria. Potrebbe essere un piccolo progetto che rimane tale, ma potrebbe anche

crescere e diventare estremamente complesso, coinvolgendo non solamente i servizi SaaS e soluzioni mobili ma anche i sistemi legacy già presenti on-premise. Per questo si dovrebbe utilizzare un iPaas che, potenzialmente, permetta un integrazione di entrambi gli ambiti.

- Fornire una alta scalabilità con una latenza bassa.
  Risulta cruciale essere in grado di gestire il processamento parallelo. Queste performance possono essere ottenute tramite una architettura cosiddetta a microservizi. Lo stile architetturale a microservizi è un approccio allo sviluppo di una singola applicazione come insieme di piccoli servizi, ciascuno dei quali viene eseguito da un proprio processo e comunica con un meccanismo snello, spesso una HTTP API. Per questo deve essere preferito un iPaaS che sia basato sui microservizi.

2.4.2 Alcuni vendor iPaaS

Non tutti i vendor iPaaS includono l'integrazione IoT nel proprio portfolio, focalizzandosi solo su applicazioni SaaS o locali e, anche se molto validi, non ci interessano nella trattazione di questa tesi. Alcuni vendor iPaaS che includono anche lo IoT tra le proprie integrazioni sono:

Snaplogic L'azienda nasce nel 2009 da più investitori privati. Utilizza dei connettori chiamati Snaps. Sfrutta, se esistenti le API dei vendor SaaS, utilizzando gli Snaps per incapsulare interfaccia e ottimizzare all'accesso delle API, rendendole di più semplice utilizzo ed eliminando all'utente problemi dovuti alla modifica remota di API. Snaplogic garantisce una bassa latenza nell'integrazione iPaaS/sistemi on-premise tramite delle pipeline sempre attive che rispondono alle richieste esterne. Queste pipeline sono distribuite in modo da poter essere eseguite in maniera concorrente. La distribuzione viene fatta rispettando la gravità dei dati, quindi avendo due possibili luoghi dove rilasciare il proprio kernel di esecuizione dell'integrazione(Snaplex), nel cloud o on-premise, in base a dove è la richiesta. E' un iPaaS che utilizza per comunicare solo messaggi Json, trasformando tutto quello che manipola in messaggi di questo tipo. Come interfaccia per gli utenti utilizza tre moduli in HTML5 residenti nel Cloud che sono: Designer, per creare i propri workflow di integrazione, Manager, dove si controllano le performance

e la sicurezza dei propri workflow, e Monitoring Dashboard, per controllare consumi e stato della propria infrastruttura iPaaS.

MuleSoft E' una società creata più di una decina di anni fa ed ha cominciato producendo Enterprise Service Bus (ESB), che può essere considerato l'antenato utilizzato in locale dell'iPaaS. Il core della connettività dell'iPaaS si chiama Anypoint Exchange. l'iPaaS di Mulesoft utilizza una adesione alle specifiche RAML per costruire le proprie API RESTful, che poi possono essere rilevate ed utilizzate tramite Anypoint Exchange. Per quello che non ha invece API, come database, FTP o SAP, mette a disposizione dei connettori customizzati. Per garantire una bassa latenza nell'integrazione iPaaS/sistemi on-premise permette una connessione tramite VPN dei dati residenti nel cloud e i data center locali. In più Anypoint Exchange utilizza il caching per evitare continue richieste alle proprie applicazioni on-premise e gestendo le richieste tramite un sistema di code chiamto Anypoint MQ che permette di servire le varie richieste mano a mano che se ne ha la possibilità.

Microsoft Azure IoT Suite Microsoft Azure IoT Suite nasce come costola di Microsoft Azure, che è una raccolta di servizi cloud di analisi, elaborazione, database, applicazioni per dispositivi mobili, rete, archiviazione e Web. Azure utilizza, se nativamente esistenti, le API RESTful fornite dai vari servizi SaaS, utilizzando però Logic App per orchestrare le varie connessioni e fornendo eventualmete alcuni connettori per i servizi che non possiedono API. Per garantire una bassa latenza nell'integrazione iPaaS/sistemi on-premise viene rilasciato direttamente sul sistema locale Azure Hybrid Connection, utilizzato quando in uno dei workflow è necessaria una integrazione con una applicazione on-premise. Vengono utilizzati inoltre dei pattern asincroni per cercare di compensare i ritardi di risposta del cloud ma se serve invece sincronizzazione vengono creati dei webhook, utilizzabili quando necessari. Tramite l'Advanced Analytics, cioè algoritmi di analisi avanzate dei dati ottenuti dai vari dispositivi, e l'apprendimento automatico si possono ottenere analisi approfondite dei dati e creare dei progetti di manutenzione predittiva del sistema.

IFTTT E' un iPaaS che fa della semplicità e del gran numero di servizi inclusi il suo punto di forza. Non ha integrazioni diverse da quelle SaaS, quindi probabilmente non è la soluzione adatta per una azienda con tutte

le complessità ed i suoi servizi e software on-premise esistenti, ma permette integrazioni con IoT molto semplicemente. Questa funzionalità risulta essere facilemte applicabile a quasi tutti gli iPaaS, che utilizzano API per i loro bisogno di integrazioni, proprio per questo una analisi più approfondita sul suo funzionamento verrà fatta nel capitolo successivo.

Capitolo 3

IFTTT:If This Than That

Figura 3.1: Logo di IFTTT

3.1 Introduzione

Come visto quindi esistono diverse offerte di iPaaS. La maggior parte di questi sono delle piattaforme omnicomprensive che forniscono integrazione a tutti i livelli. I maggiori produttori dei servizi iPaaS sono spesso software house che ovviamente cercano di fornire servizi per le aziende, in quanto naturale core business dell'integrazione, vista la complessità delle interazioni al suo interno.
Gartner definisce gli utilizzatori del modello di integrazione in base al proprio spazio all'interno del modello:

- specialisti dell'integrazione, che fanno quello di lavoro e quindi il loro compito è, ad esempio, creare API che permettano di estrapolare dati in modo che possano essere usati proficuamente dalla linea di business, considerando ovviamente anche tutto ciò che ha a che fare con questo. Si prendono carico anche che, ad esempio, i dati siano coerenti, che gli utenti non possano far crashare il sistema o rovinare i dati, limitare e bilanciare gli accessi etc.

- integratori ad-hoc, che fanno parte della linea di business ed utilizzano saltuariamente l'integrazione. Solitamente il loro interesse è nel poter accedere ai dati per poterli utilizzare in una qualche forma di workflow decisionale.

- integratori cittadini, utenti non tecnici che generalmente sono , molto spesso, esclusi da ogni forma di integrazione.

Proprio a questi utenti, che sono esclusi dall'integrazione aziendale ma che usano comunque mail e smartphone anche al di fuori del luogo di lavoro, che hanno termostati connessi a Internet e tracker della propria condizione fisica, si rivolge IFTTT, un iPaaS gratuito e molto intuitivo.
IFTTT cerca di rendere più efficienti, salutari ed ecologiche le nostre vite, sia all'interno che al di fuori dell'azienda, tramite piccole automazioni, collegando tra loro i servizi e gli strumenti che quotidianamente ci circondano ed utilizziamo, per permetterci di avere una vita, digitale nel caso di servizi come mail o social network oppure reale nel caso di oggetti connessi a Internet, più semplice.

3.2 Cosa fa

IFTTT è un iPaaS che permette di collegare assieme tutta una serie di servizi web da parte di terzi usando la semplice regola "se succede questo fai quest'altro". Il concetto fondamentale del servizio ruota attorno ad alcune parole chiave, che permettono in maniera diretta, di inquadrare meglio le potenzialità del servizio.
La prima parola chiave è "canale". Ogni singolo prodotto o servizio è a tutti gli effetti un canale. Ad esempio Gmail è un canale, come lo è Twitter, come lo sono però anche i prodotti WeMo (prese di corrente collegabili alla rete) o le luci Philips Hue.

Poi esistono le "ricette" che sono il cuore di IFTTT e principalmente sono quelle che legano due canali secondo la semplice regola "Se succede questo in questo canale fai quest'altro in quest'altro canale".
Ogni canale appunto può essere presente come THIS, e quindi avere tutta

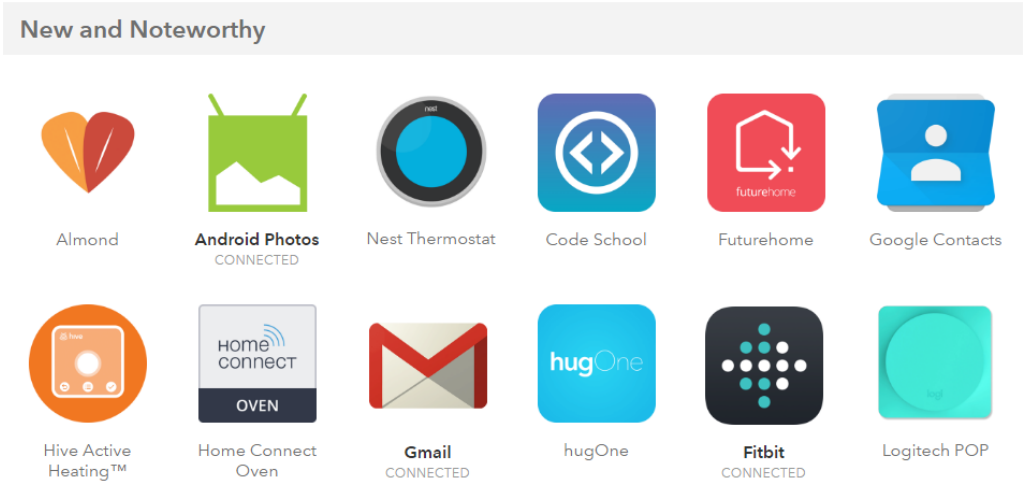

Figura 3.2: Solamente alcuni canali di IFTTT tra gli oltre 300 disponibili

una serie di "trigger" che possono far scattare la seconda parte della ricetta , oppure essere presenti come THAT, e quindi far eseguire una "azione" sul canale corrispondente.
La cosa più vicina a delle variabili sono gli "ingredienti", presenti in ogni trigger e sceglibili in base al canale su cui si sta lavorando, che vengono passate all'azione quando la ricetta viene eseguita. Alcuni esempi possono essere:
Un trigger di gmail che permette di fare qualcosa quando si riceve una mail da uno specifico indirizzo mail. L'indirizzo mail è un ingrediente.
In Twitter un trigger che viene attivato ogni volta che un utente specifico fa un nuovo tweet. L'utente è l'ingrediente.

Questo è principalmente il funzionamento nell'essenza di IFTTT e le cose che diventano le sue caratteristiche peculiari sono principalmente due:
1. E' assolutamente alla portata di chiunque, anche dell'utente senza neanche la minima concezione di programmazione quindi SEMPLICITA'

2. Consente di unire tra loro canali completamente diversi, originariamente non pensati per essere usati in combinazione tra loro, aumentando così esponenzialmente l'utilità di ogni singolo canale quindi INTEGRAZIONE

La semplicità è quello che probabilmente lo ha portato ad essere un servizio così diffuso. Per creare una ricetta, come detto, non si deve conoscere assolutamente nessun tipo di linguaggio di programmazione e l'unico concetto che serve capire è quello di condizione if, mutuabile semplicemente dall'esperienza quotidiana senza doverlo necessariamente associare a nulla di informatico. La creazione avviene tramite un semplice wizard in cui la maggior parte delle scelte sono effettuabili tramite dei click sulle icone dei canali e sulle varie condizioni / azioni. Le condizioni non sono mai più lunghe di un paio di righe e sempre assolutamente chiare ed esplicite. L'unica cosa un minimo meno triviale sono gli ingredienti, a tutti gli effetti variabili, che possono essere utilizzate come confronto se contenute nella parte di THIS o come valore se nella parte di THAT. Molto spesso gli ingredienti saranno ovvi in base al canale prescelto (es. mail arrivate su gmail con una certa parola, tracce caricate con una certa parola nel titolo su soundcloud, un tweet da un utente specifico etc.) , ancora più spesso gli ingredienti non saranno nemmeno presenti. Oltre a poter creare ricette da zero si può anche utilizzare la sterminata quantità di ricette già create dalla comunità e condivise per potter essere usate da chiunque. Oltre che essere già pronte per essere usate sono anche già divise in base all'ambito di utilizzo, andando dagli usi più prosaici, come aumentare la produttività o risparmire sulle bollette, a quelli più strani, come organizzare il proprio matrimonio o "ricette dallo spazio profondo" tramite servizi messi a disposizione dalla NASA.

L'integrazione tra così tanti servizi diversi (323 nel momento in cui scrivo) lo rende il middleware di integrazione cloud di tipo SaaS più vasto. Inizialmente era stato pensato per unire semplicemente servizi web di terze parti, rendendo l'esperienza dell'utilizzo quotidiano dei vari servizi online meno frammentaria e soprattutto estramamente più funzionale. Ma la vera rivoluzione si ha nel 2012, quando IFTTT entra nel regno di IoT, includendo tra i propri canali anche le device per la domotica Belkin Wemo, luci e prese di corrente collegabili a Internet, e quindi uscendo dal mondo puramente astratto del web per cominciare ad interagire con il mondo fisico. Ora la lista dei canali è, come detto, molto vasta ed unisce indifferentemente oggetti e servizi, chiamandoli tutti canali, la qual cosa tende ancora di più a rendere

effimera la linea di separazione tra le due. Ovviamente adesso fare parte del vasto gruppo dei canali di IFTTT è un notevole aumento del valore del prodotto, quindi sia nelle scatole di imballaggio che nelle homepage dei siti campeggia in primo piano la piena compatibilità con il servizio. Si è arrivati addirittura al punto in cui molti oggetti hanno addirittura dei software di controllo proprietari abbastanza scadenti e non particolarmente rifiniti, ma delle API, quindi utilizzabili da IFTTT, molto più curate, delegando a tutti gli effetti il controllo del proprio prodotto a IFTTT stesso e a uno qualunque degli altri canali Inoltre in ogni canale si può trovare, sottoforma di lista a piè pagina, una serie di altri canali con cui, per funzionalità scopi o modalità di utilizzo, viene garantito un ottimo funzionamento con il canale corrente. Con questo metodo, semplice e non invasivo, si incentiva alla scoperta e all'integrazione da parte degli utenti di nuovi servizi di cui magari prima non si conosceva nemmeno l'esistenza.

Proprio grazie a queste due caratteristiche (semplicità e integrazione totale) si ha tra le mani un servizio che permette di ottenere una task automation eccellente in svariati ambiti. Basta infatti fare qualche ricerca in rete per vedere quanti risultati ci mostrino varie collezioni di ricette atte ad ottenere un incremento della produttività: dal postare su tutti i social contemporaneamente per coloro che li usano come principale mezzo di marketing a segnare su google calendar gli appuntamenti e le scadenze schedulate tramite piattaforme di collaboration online, salvare le proprie ricevute su cloud oppure tenere su un foglio di calcolo le ore lavorate. Tutto questo in maniera quasi o totalmente automatica. Ma le ricette che più ci interessano sono quelle che entrano nell'ambito dello IoT e il guadagno non si riduce solo a un mero aumento di produttività ma a un miglioramento della vita in molti dei suoi ambiti che non siano solo quelli lavorativi: risparmio di denaro, ambienti e stili di vita più salutari fino ad arrivare alla sicurezza. Ad esempio è possibile programamre un termostato Honeywell per settarsi in modalità economica quando Life360 mi dice che non c'è nessuno a casa, dare un comando ad Alexa che mi permetta di mettere in funzione il mio sistema di sicurezza Piper, ricevere un reminder sullo smartphone che mi ricorda di andare a letto prima se il tracker Fitbit ha rilevato una notte insonne, oppure spegnere l'umidificatore tramite WeMo Switch se Awair rileva troppa umidità nell'aria. Ovviamente questi sono solo alcuni esempi di ciò che si può fare e la maggior parte degli oggetti IoT compiono solo semplici funzioni basandosi su pochi parametri e non utilizzando la miriade di dati

che la rete può fornire ma è comunque un ottimo inizio per l'integrazione di IoT nella vita quotidiana

3.3 Come lo fa

L'utilizzo del servizio a livello utente è semplicissimo, basta sottoscriversi a IFTTT per cominciare a creare ricette o ad usare quelle altrui. Quando una ricetta contiene un canale mai utilizzato viene richiesta la connessione da parte dell'utente a quel servizio. Da quel momento in poi il canale sarà legato al proprio account IFTTT e quindi liberamente utilizzabile per tutte le future ricette. Andando a cercare invece come tecnicamente IFTTT realizza le varie integrazioni tra canali non si riesce a trovare molta documentazione esplicita, per cui risulta più semplice emulare quello che IFTTT fa con un esempio pratico.

Per analizzare il funzionamente tecnico di IFTTT come iPaaS per IoT useremo un esempio nel quale faremo esattamente quello che IFTTT permette di fare. Gli esempi ovviamente potrebbero essere innumerevoli ma, per dare un' idea generale dell' uso e della connessione dei vari servizi, basta vederlo una volta, ricordandoci come moltissimi iPaaS funzionino con il medesimo metodo, ovvero sfruttando le API native dei servizi SaaS. Ovviamente le procedure generali rimarranno molto simili anche cambiando i servizi che si dovranno legare mentre le procedure specifiche varieranno da canale a canale in base alle API messe a disposizione.
La ricetta che riprodurremo sarà la seguente:

Figura 3.3: Ricetta IFTTT con Fitbit e Gmail

Il primo canale è Fitbit, una marca di dispositivi indossabili che permettono di monitorare l'attività fisica, il battito cardiaco e la qualità del sonno, mentre il secondo è Gmail, il noto servizio SaaS di mail.
La ricetta prevede che, se raggiunti un certo numero di passi decisi come goal giornaliero, venga mandata una mail ad un receiver ben preciso per ricordare il risultato conseguito.
Questa è una minimizzazione di quello che si può fare, in quanto si potrebbe fare una interazione da IoT a IoT, quindi sempre rimanendo nell'ambito degli oggetti connessi, ma a livello di procedura cambia poco quindi ci limiteremo ad una banale mail.

Per poter creare l'automazione tra i due canali bisogna prima di tutto avere accesso ai dati dell'utente di quel servizio, e l'unico modo per avere l'accesso tramite una applicazione esternaè utilizzare le API messe a disposizione. Quindi nel caso specifico bisogna usare le API di FITBIT per leggere se l'utente ha raggiunto o meno l'obiettivo giornaliero. Di questo dato bisogna fare un check periodico per controllare se la condizione dell'IF è stata soddisfatta. In caso positivo si procede utlizzando le API di GMAIL per spedire la mail con il messaggio di successo.

3.3.1 API

Ruoli Per chiarezza elencheremo di seguito i ruoli che prenderanno parte al processo di utilizzo:

1) APPLICAZIONE DI TERZE PARTI(CLIENT): è l'applicazione che stiamo scrivendo (quindi tipo ifttt) ed è quella che vuole utilizzare le informazioni dell'account dell'utente. Prima di poterlo fare ha bisogno dell'autorizzazione dell'utente stesso. Da ora in poi verrà chiamata APPLICAZIONE

2) API(RESOURCE SERVER): è il server che ci mette a disposizione le API per poter accedere alle informazioni dell'utente. Da ora in poi verrà chiamato web server, mentre quello che fornisce verrà chiamato SERVIZIO.

3) UTENTE(RESOURCE OWNER): è l'utente che dà la possibilità di accedere a parte delle proprie informazioni oppure ad utilizzare un servizio in propria vece sui vari web server. Da ora in poi verrà chiamato UTENTE.

OAuth 2.0 Le API messe a disposizione da fitbit e google sono totalmente libere (nel senso non sono un servizio a pagamento tranne che per alcuni casi specifici) quindi utlizzabili. Questo non significa che chiunque possa usarle indiscriminatamente.
Proprio per questo è stato creato un protocollo standard di controllo e autenticazione chiamato OAuth 2.0. Esso è una evoluzione dell'OAuth 1.0a che, oltre ad essere più semplice, risulta anche più sicuro, permettendo alle applicazioni che lo usano come metodo di autorizzazione di non dover conoscere, e quindi mantenere memorizzate, username e password dell'utente per accedere al servizio fornito dal web server. L'autorizzazione con credenziali è sostituita da una serie di token che garantiscono l'accesso e la riservatezza.

Per utilizzare l'OAuth 2.0 in una applicazione web, prima bisogna creare le credenziali della nostra applicazione registrandosi al web server. Quando l'applicazione ha bisogno di accedere alle informazioni usando le API, l'applicazione redirige gli utenti al web server OAuth 2.0. Il web server autentica l'utente e ottiene il consenso dall'utente per l'applicazione a poter usare i dati dell'utente. Il web server OAuth 2.0 redirige l'utente all'applicazione assieme ad un codice di autorizzazione monouso. L'applicazione scambia questo codice per un token di accesso. Finalmente, l'applicazione può utilizzare il token di accesso per accedere alle API del web server. Questo token è tutto quello che serve alla nostra applicazione per accedere alle API del web server.

Tutto questo si può riassumere dividendo in tre parti l'utilizzo delle API:
1)Creazione
2)Autenticazione
3)Utilizzo

CREAZIONE

Prima di cominciare il processo di OAuth, bisogna registrare l'applicazione al servizio offerto dal web server. Quando si fa la registrazione dell'applicazione ai due servizi (fitbit e gmail/google) bisogna inserire alcune informazioni di base tra cui nome, sito web, logo... In aggiunta bisogna inserire un URI di redirezione per redirigere gli utenti che useranno l'applicazione tramite applicazione web-server (ma potrebbe essere anche una applicazione browser o mobile).

Redirezione dell'URI I web server, solitamente, permettono la redirezione degli utenti solamente a URI registrati, utilizzando questa accortezza come una prima forma di sicurezza. Per ogni redirezione http ad un URI è consigliata la protezione tramite TLS, in questo modo il web server farà redirezioni ad URI che cominciano con "https", prevenendo l'intercettazione del token durante il processo di autorizzazione grazie alla crittazione dei dati.

Client ID and Secret ID Dopo aver registrato la propria applicazione, si riceverà un client ID e un secret ID. Il client ID è una informazione considerata di pubblico dominio e verrà usata per costruire l'URL di login dell'applicazione. Il secret ID invece deve essere mantenuta privata a tutti i costi.
Nel nostro caso creiamo quindi due applicazioni con lo stesso nome, una registrata su Google e l'altra su Fitbit.

Prima quella su Google creata abilitando le API di gmail per l'invio di email come mostrato in figura 3.2

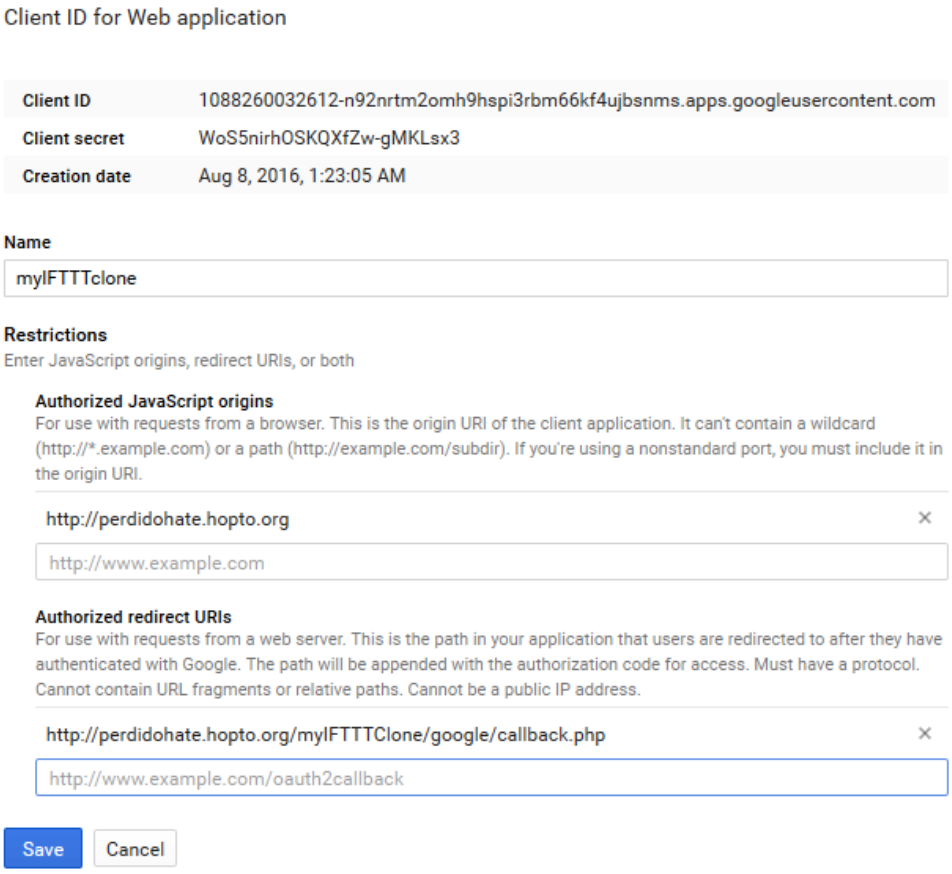

Figura 3.4: Registrazione della applicazione in Google

Poi quella di FitBit, questa volta creata invece con la sola abilità di lettura come mostrato in figura 3.3

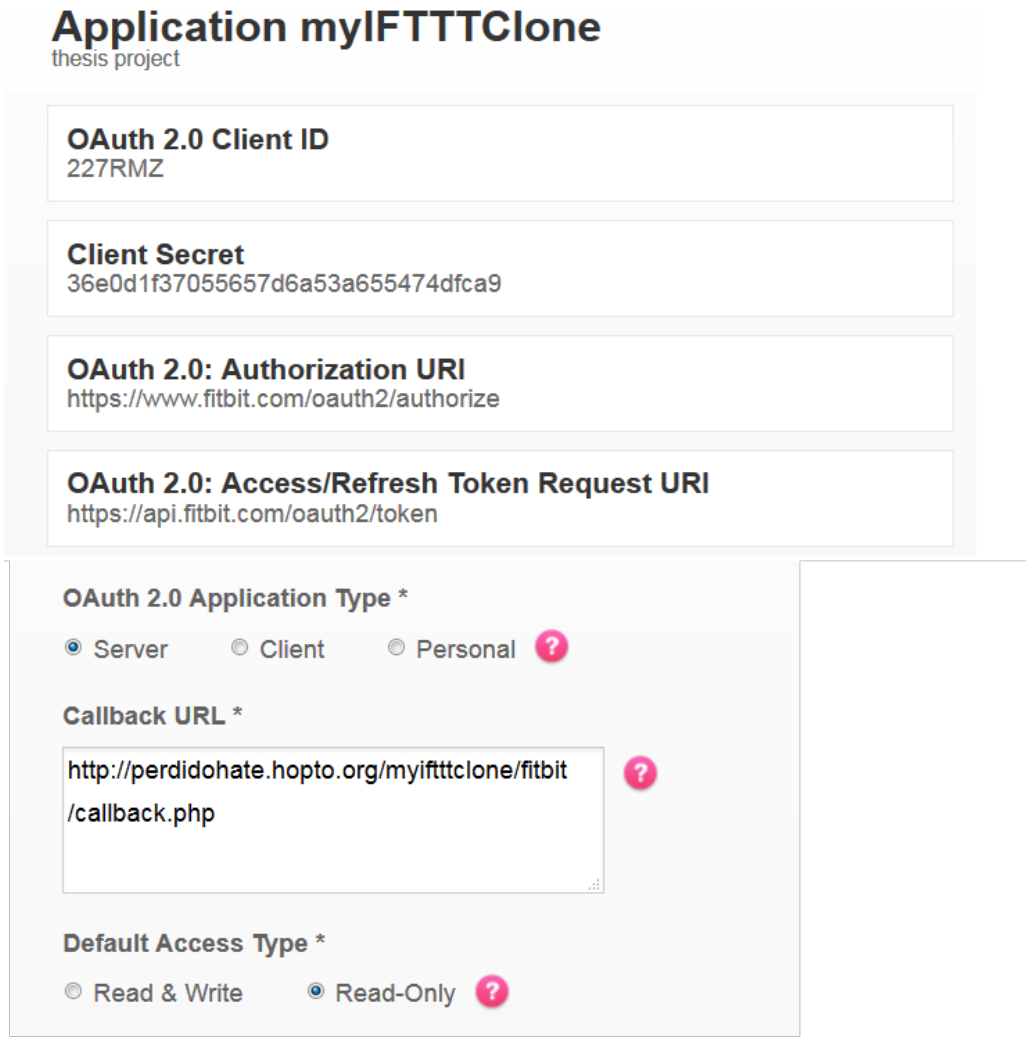

Figura 3.5: Registrazione della applicazione in Fitbit-account

AUTENTICAZIONE

Esistono vari modi per ottenere l'autorizzazione,generalmente scelto in base al tipo di applicazione sviluppata. Quello utilizzato da noi, avendo un server che contiene la nostra applicazione, sarà del tipo chiamato "Authorization Code".

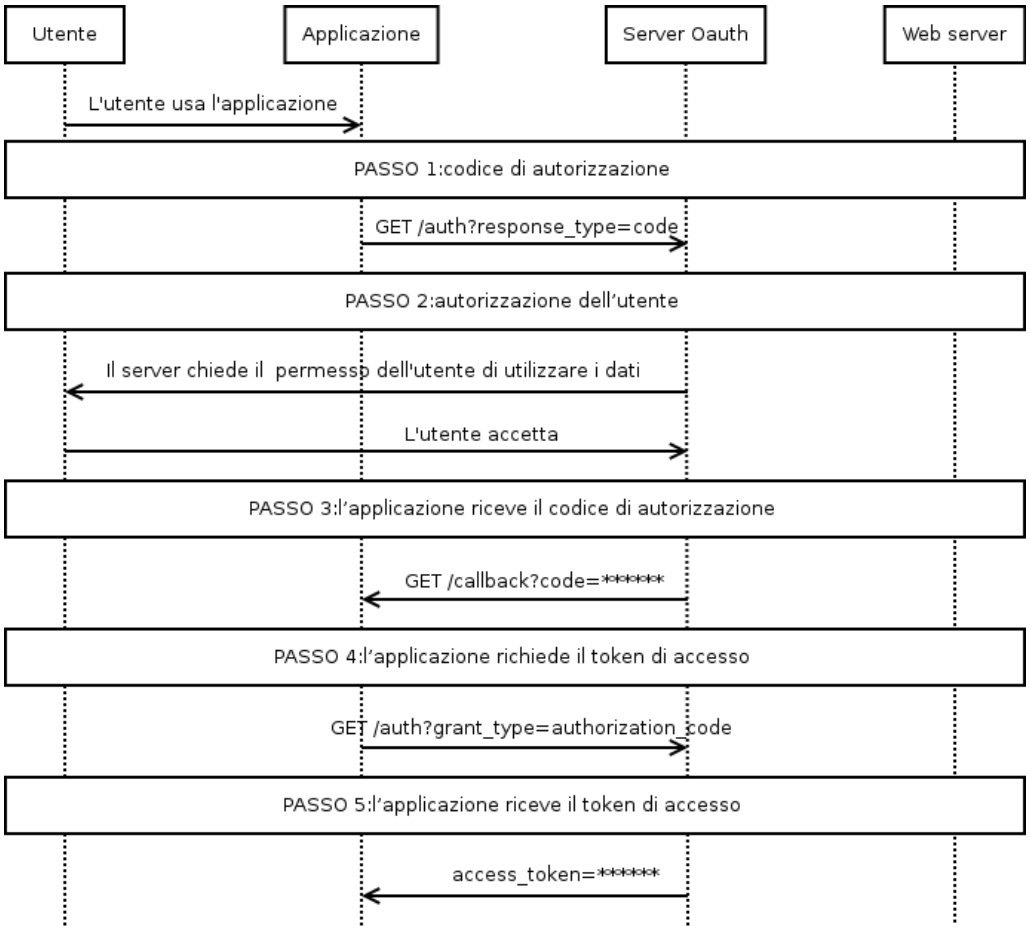

Figura 3.6: Schema della sequenza Oauth2

Il processo si divide in 5 punti:

PASSO 1:codice di autorizzazione Il primo passo del processo è ottenere l'autorizzazione dall'utente. Questa autenticazione non è altro che una richiesta al web server fatta tramite http. Quindi a tutti gli effetti è solamente un link che punta all'authorization endpoint del server web e gli passa almeno 4 parametri:
- client_id= è l'id identificativo dell'applicazione
- redirect_uri= è l'URI a cui deve passare l'authorization code il web server una volta che l'utente ha accettato di permettere all'applicazione di usare i propri dati
- response_type=code specifica che la nostra applicazione vuole l'authorization code
- scope= indica a quali dati vuole avere accesso l'applicazione

La richiesta fatta a google risulta così

https://accounts.google.com/o/oauth2/auth?
response_type=code&
redirect_uri=http%3A%2F%2Fperdidohate.hopto.org%2Fmyifttttclone
%2Fgoogle%2Fcallback.php&
client_id=1088260032612-n92nrtm2omh9hspi3rbm66kf4ujbsnms.apps.
googleusercontent.com&
scope=https%3A%2F%2Fwww.googleapis.com%2Fauth%2Fgmail.send&
access_type=offline&
approval_prompt=auto

Mentre la richiesta fatta a fitbit risulta così

https://www.fitbit.com/oauth2/authorize?
client_id=227RMZ&
redirect_uri=http%3A%2F%2Fperdidohate.hopto.org%2Fmyifttttclone
%2Ffitbit%2Fcallback.php&
scope=activity&
response_type=code

PASSO 2:autorizzazione dell'utente A questo punto il web server mostra la pagina di richiesta delle autorizzazioni come mostrato in figura 3.5

per Google e in figura 3.6 per Fitbit

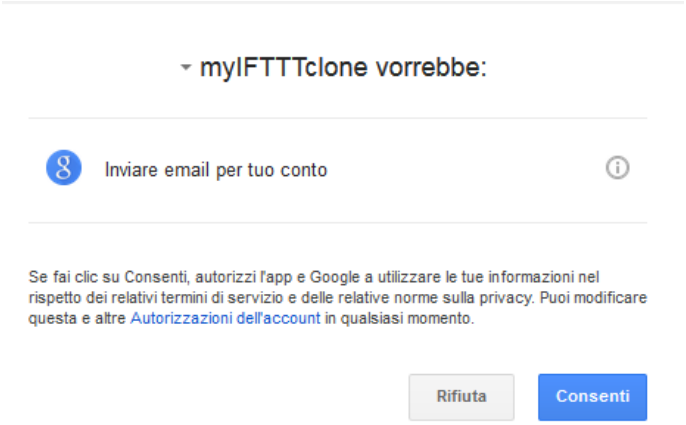

Figura 3.7: Richiesta di utilizzo all'utente per Google

Figura 3.8: Richiesta di utilizzo all'utente per Fitbit

PASSO 3:l'applicazione riceve il codice di autorizzazione Quando l'utente clicca "consenti", il web server genera un codice di autorizzazione che viene rimandato al callback URI sempre tramite request http

Ecco quello che viene generato per Google:

http://perdidohate.hopto.org/myifttttclone/google/callback?
code=4/kTMjm6k_7mZqW50gCjk-hXG-30rm6iAsxO8wEs0X_n8#

Mentre quello generato per Fitbit risulta essere:

http://perdidohate.hopto.org/myifttttclone/fitbit/callback.php?
code=d919f675c8006799cc7818f5652f9707858d9535#_=_

PASSO 4:l'applicazione richiede il token di accesso A questo punto la nostra applicazione fa l'ennesima richiesta http in cui si scambia il codice di autorizzazione con un token di accesso.
Esso di solito viene fatto tramite POST, uno dei possibili metodi http.
I parametri necessari che devo passare al web server sono:
- client_id= è l'id identificativo dell'applicazione
- client_secret= è la password della nostra applicazione
- redirect_uri= è l'URI a cui deve passare l'access token il web server
- grant_type=authorization_code specifica che la nostra applicazione vuole scambiare l'authorization code con un access token
- code= è l'authorization code che il web server ci ha fornito come risposta al punto precedente

La richiesta a google

https://accounts.google.com/o/oauth2/auth?
code=4%2FwRbhe1um37weZ_5EhPVI9AKW8HRjQa3gYQnCaovPq4g&
redirect_uri=http%3A%2F%2Fperdidohate.hopto.org%2Fmyifttttclone
%2Fgoogle%2Fcallback.php&
client_id=407408718192.apps.googleusercontent.com&
client_secret=**********&

scope=&
grant_type=authorization_code

La richiesta a fitbit

https://api.fitbit.com/oauth2/token?
client_id=227RMZ&
grant_type=authorization_code&
redirect_uri=http%3A%2F%2Fperdidohate.hopto.org%2Fmyifttttclone
%2Ffitbit%2Fcallback.php&
code=a7573f6ce722073103c39dd34375bc16ec2deded

Come si può notare entrambe le richieste contengono tutti i dati, anche quelli già precedentemente inviati, e non solamente il code token.
Questo per avere una condizione stateless, cioè ogni transazione può essere fatta indipendentemente perchè contiene tutti i dati, come detto da uno dei paradigmi del REST.

PASSO 5:l'applicazione riceve il token di accesso Il web server risponde con un access token in formato json che finalmente ci permette di avere accesso alle API

Il token di google è creato in questo modo

{"access_token":"ya29.CjA6AzVqTUV8-ZO76C0xRHb2CQ1d50HOt7xaEO
r8y7GlaUUdLHXRafh_1vm782bWyKI",
"token_type":"Bearer",
"Expires_in":3589,
"Created":1470746492
}
Mentre nel token di fitbit c'è anche il refresh token, usato in caso di scadenza dell'originale

{"access_token":"eyJhbGciOiJIUzI1NiJ9.eyJzdWIiOiI0VFJHQk4iLCJhdW
QiOiIyMjdSTVoiLCJpc3MiOiJGaXRiaXQiLCJ0eXAiOiJhY2Nlc3NfdG
9rZW4iLCJzY29wZXMiOiJyYWN0IiwiZXhwIjoxNDcwOTAwNzM4LC

JpYXQiOjE0NzA4NzE5Mzh9.LGi_dxjszwDs1-uaNeJ-yxmOA4u2BykdA
9HxtCazc1c",
"expires_in":28800,
"refresh_token":"ea4a8663e4514b3750e94f13e5b26ee8d2d3de7ab618c0bdee7
6fb8d4a883e0b",
"scope":"activity",
"token_type":"Bearer",
"user_id":"4TRGBN"
}

UTILIZZO

A questo punto, avendo avuto l'autorizzazione sia dall'utente che da entrambi i web server, posso fare le mie chiamate alle API dei due servizi.

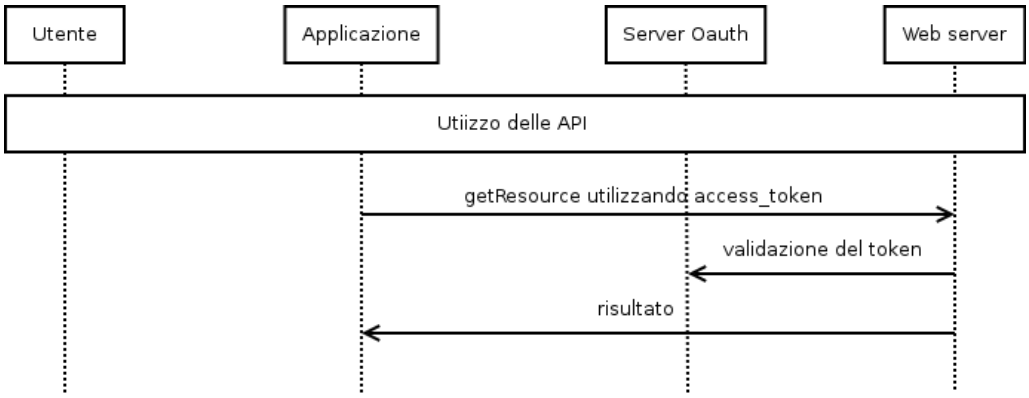

Figura 3.9: Schema della sequenza di scambio credenziali per l'uso dell API

Entrambe le API sono di tipo RESTful, il che significa che sono conformi ai paradigmi REST.

I paradigmi REST sono 5 più uno opzionale:

Client-Server si deve avere una netta divisione tra i due ruoli per cui, ad esempio, il client non si dovrà mai preoccupare di salvare informazioni

mentre il server non si dovrà mai preoccupare dell' interfaccia grafica.
La comunicazione tra i due avviene tramite interfaccia.

Uniform Interface essendo client e server indipendenti devono comunicare sottostando alle direttive di un interfaccia omogenea

Steteless nella comunicazione client-server il client ha sempre tutte le informazioni che servono per richiedere quel servizio, quindi senza doversi appoggiare al server per mantenere dati sullo stato della sessione.

Cacheable i client possono salvare in cache alcune risposte del server per evitare di doverne fare richiesta in futuro.
Per ogni risorsa deve essere definito esplicitamente se sia possibile metterla in cache o no, per evitare l'utilizzo di dati non aggiornati da parte del client.

Layered system possono esistere layer intermedi di server che hanno alcune funzioni specifiche che devono essere totalmente trasparenti rispetto al client, il quale deve essere struttarato come se parlasse sempre direttamente con il server finale.

Code on Demand(opzionale) a volte i server possono estendere i client con pezzi di codice che vengono trasferiti direttamente al client ed eseguiti

I Servizi REST sono catalogati come ROA (Resource Oriente Interface) in quanto il concetto centrale di questo tipo di architettura è la risorsa. Infatti per ottenere i dati sulle risorse si fa una ben specifica richiesta http in cui sia l'URI (chiamato endpoint) che il metodo con cui lo si chiama (GET,POST,PUT,DELETE etc.) sono influenti e si riceve indietro un messaggio JSON o XML con le informazioni richieste, le quali sono una rappresentazione di quella risorsa, intesa come descrizione dello stato corrente

della risorsa stessa. Ovviamente, cambiando gli endpoint e i metodi, non è solo possibile avere una lettura dei dati ma anche una scrittura o un utilizzo del servizio.

Nel nostro caso specifico dovremo leggere i dati da Fitbit per sapere se il goal giornaliero è stato raggiunto ed utilizzare il servizio di invio di Gmail per inviare una email. Per leggere i dati da Fitbit, e quindi vedere se il nostro utente ha raggiunto l'obbiettivo previsto, dobbiamo utilizzare il seguente metodo ed endpoint:

GET https://api.fitbit.com/1/user/[user-id]/activities/date/[date].json

Come parametri si devono inserire [user-id] che nel nostro caso sarà "-" in quanto indica l'utente connesso attualmente e [date] nel formato yyyy-MM-dd.
Quindi dopo aver effettuato il parsing della data attuale si invierà la richiesta, ad esempio

GET https://api.fitbit.com/1/user/-/activities/date/2016-08-09.json

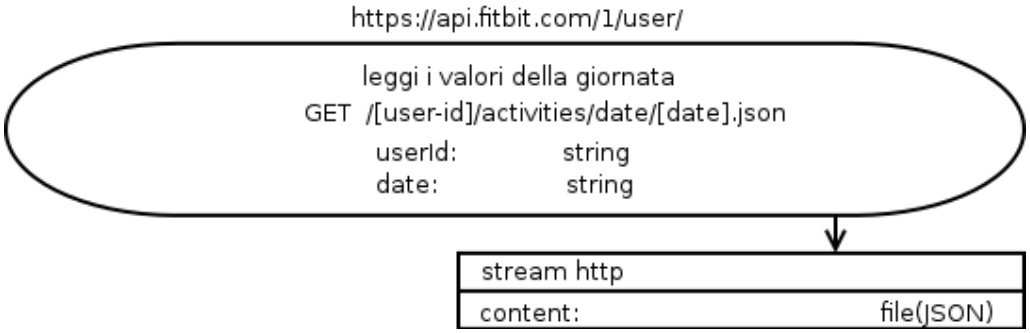

Figura 3.10: Rappresentazione UML dei dati in lettura da Fitbit. Non avendo una classe fornita da Fitbit stessoper la lettura dei dati, si riceve direttamente lo stram http

Ecco un piccolo estratto del codice:

```
$today = date("Y-m-d");
```

```
$api_url = 'https://api.fitbit.com/1/user/-/activities/    >>
>> date/'.$today.'.json';
$header = 'Authorization: Bearer ' . $access_token;
$options = array(
    'http' => array(
        'method' => 'GET',
        'header' => $header,
        'ignore_errors' => true
    )
);

$context = stream_context_create($options);

$res = file_get_contents($api_url, false, $context);
var_dump($res);
$activity = json_decode($res, true);
```

E la variabile $res che contiene la risposta JSON sarà fatta così:

```
{
    "activities": [],
    "goals": {
        "activeMinutes": 30,
        "caloriesOut": 2593,
        "distance": 8.05,
        "floors": 10,
        "steps": 5000
    },
    "summary": {
        "activeScore": -1,
        "activityCalories": 1035,
        "caloriesBMR": 1565,
        "caloriesOut": 2430,
        "distances": [],
        "elevation": 39.62,
        "fairlyActiveMinutes": 48,
        "floors": 13,
        "heartRateZones": [],
```

```
        "lightlyActiveMinutes": 172,
        "marginalCalories": 597,
        "restingHeartRate": 51,
        "sedentaryMinutes": 770,
        "steps": 7242,
        "veryActiveMinutes": 16
    }
}
```


A noi interessa confrontare:


```
{
    "goals": {
        "steps": 5000
    }
}
```


con


```
{
    "summary": {
        "steps": 7242,
    }
}
```


Su questi due dati bisogna fare il fantomatico IF del nostro programma di task automation. In caso affermativo, come in questo caso, bisogna inviare una mail usando Gmail. Per inviare una mail bisogna usare il seguente metodo ed endpoint:

POST https://www.googleapis.com/gmail/v1/users/[userId]/messages/send

Come parametri si deve sostituire [userId] con "me" in modo da indicare l'utente attualmente loggato, cioè colui che vuole usare la nostra applicazione.

Ovviamente mettendo Google a disposizione le proprie librerie l'invio avviene ad un livello più alto di astrazione della chiamata diretta all'url ma

il concetto rimane il medesimo.
Infatti si crea un oggetto Google_Service_Gmail al cui interno si crea la mail che, dopo le dovute conversioni, viene inviata sfruttando la funzione sendMessage, il quale non fa altro che sfruttare l'endpoint per inviare finalmente il nostro messaggio mail.

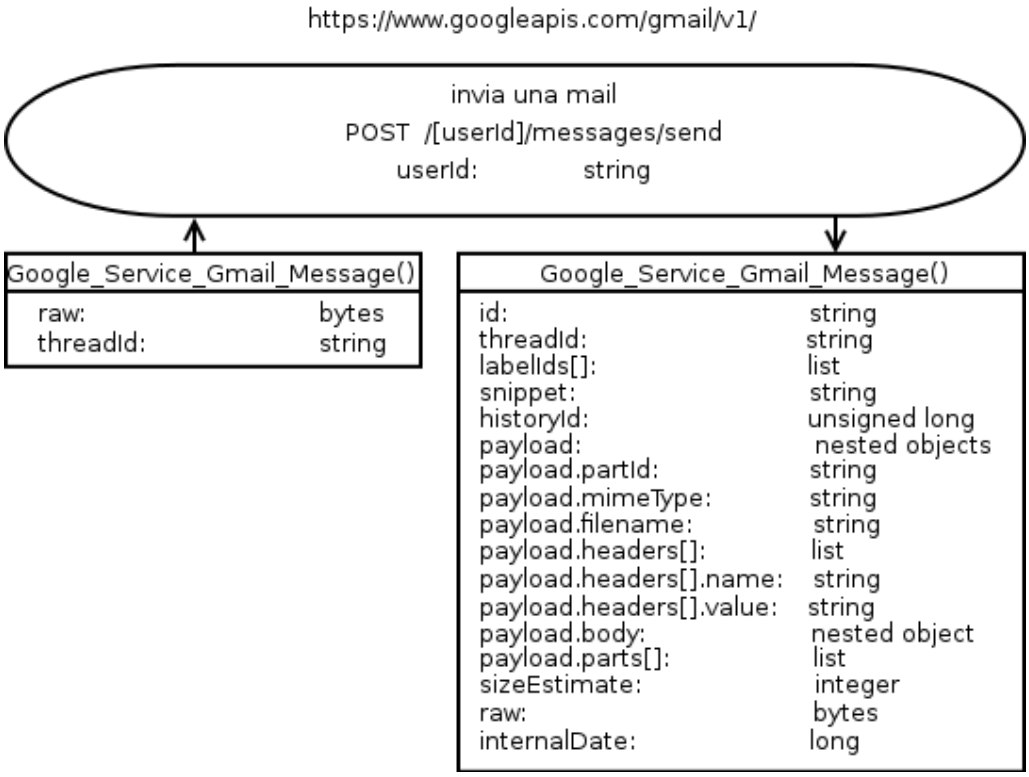

Figura 3.11: Rappresentazione UML della classe di Google che gestisce i messaggi mail

Ecco un piccolo estratto del codice:

```
$clientID = getClient();
$serviceID = new Google_Service_Gmail($clientID);
$userID = 'me';
$mail = 'From: myiftttclone <myiftttclone@noreply.com>
To: alex gavatta <perdidohate@gmail.com>
Subject: Goal reached
Date: Tue, 09 Aug 2016 12:06:06 +0100
Message-ID: <1234@iot.example>

Hai raggiunto il tuo goal giornaliero, congratulazioni!';

$mail64 = base64_encode($mail);
$mail64 = str_replace('+','-',$mail64);
$mail64 = str_replace('/','_',$mail64);\\
$messageID = new Google_Service_Gmail_Message();
$messageID->setRaw($mail64);
$results = sendMessage($serviceID, $userID, $messageID);

function sendMessage($service, $user, $message) {
    try {
        $message = $service->users_messages->send($user, $message);
        print 'Message with ID: ' . $message->getId() . ' sent.';
        return $message;
    } catch (Exception $e) {
        print 'An error occurred: ' . $e->getMessage();
    }
}
```

Ed ecco il risultato finale

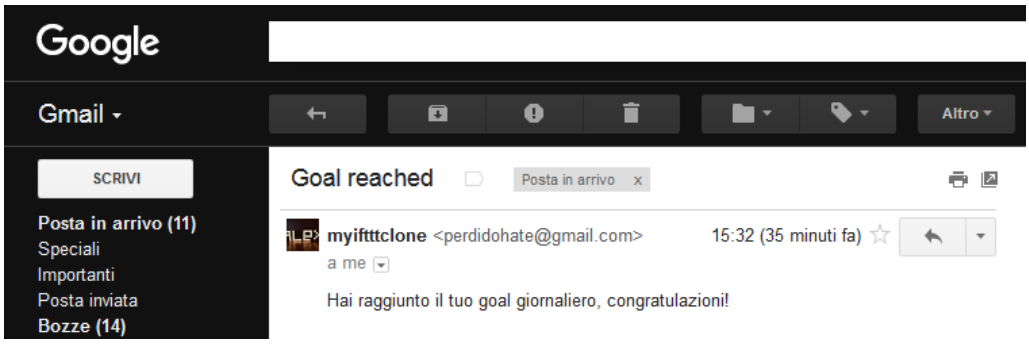

Figura 3.12: una mail di congratulazioni per ripagarti di tutta la fatica

Capitolo 4

Da oggetti a oggetti di IoT

Negli ultimi anni si è assistito ad una crescita esponenziale della cultura maker, grazie al moltiplicarsi di hardware e software dedicati.
Ormai non esistono più barriere per l'ingresso nel modo di coloro che vogliono crearsi in autonomia qualunque tipo di oggetto, sia essa una macchina a controllo numerico, uno strumento musicale personalizzato o un elettrodomestico di un qualche tipo.
Il tutto è cominciato grazie ad Arduino, piattaforma di prototipazione, a basso costo e totalmente open source, a cui ne sono seguite molte altre.
Grazie a queste piattaforme i maker possono creare e mettere in condivisione le proprie creazioni, alimentando così un circolo positivo che mantiene viva e attiva la comunità.
Le prime versioni di queste piattaforme erano ovviamente autonome e slegate, ogni creazione rimaneva ad uso e consumo di ciò che era nella prossimità del processore.
L'evoluzione naturale della cosa è stato collegarli a Internet, facendoli così divenire dei perfetti mezzi per creare degli oggetti personalizzati che appartenessero alla categoria IoT.
Oramai esiste una categoria ben precisa chiamata dello IoT chiamata "Do It Yourself".
Ovviamente poter integrare queste creazioni con un middleware che li gestisse avrebbe permesso di fare cose inizialmente impensabili.
Proprio per questo per molto tempo la comunità di maker ha chiesto agli sviluppatori di IFTTT un modo per integrare le proprie creazioni.
Ed alla fine sono stati accontentati con la creazione del canale "maker"

4.1 Canale maker

Figura 4.1: L'icona del canale MAKER

Il canale maker mette a disposizione dei webhooks, personalizzati per ogni utente, in modo da poter collegare le proprie creazioni "Do It Yourself" al resto del mondo di servizi offerti da IFTTT.
Per webhooks si intendono web request (quindi richieste http) fatte e ricevute a dei ben precisi URL.
Come detto, al canale maker è permesso sia essere un trigger, quindi essere il THIS di una ricetta, che essere usato come azione, e conseguentemente fare la parte del THAT.
Quindi, ogni oggetto che sia in grado di fare o ricevere una richiesta http, diviene perfettamente integrabile con il sistema IFTTT.

4.2 MKR 1000

Tratto dal sito www.arduino.cc:
"MKR1000 è una scheda che combina la potenza di processore alla capacità di connessione Wi-Fi. E' la soluzione ideale per i makers che vogliono

implementare dei progetti IoT".
Per scheda viene sottointeso un processore programmabile con dei pin di input e output, ai quali si possono attaccare sia sensori che attuatori.
Essa è una scheda che ricade nella categoria multi-pourpose, quindi adatta a una varia tipologia di scopi.
Le logiche da implementare, sia a livello computazionale che di controllo dei pin, vengono caricate tramite seriale e scritte in linguaggio C.
Esistono ovviamente librerie che permettono estensioni di utilizzo in ambiti specifici.
Proprio tramite delle librerie si può, ad esempio, utilizzare la connettività del chip Wi-Fi inserito nativamente nella scheda.
Grazie a questo la scheda può essere connessa a Internet e, fatto per noi importante ed indispensabile, può fare e ricevere richieste http.

4.3 Esempio this

Nel momento dell'iscrizione al canale maker viene generata una parola chiave. Questa parola chiave viene utilizzata per creare l'URI personalizzato

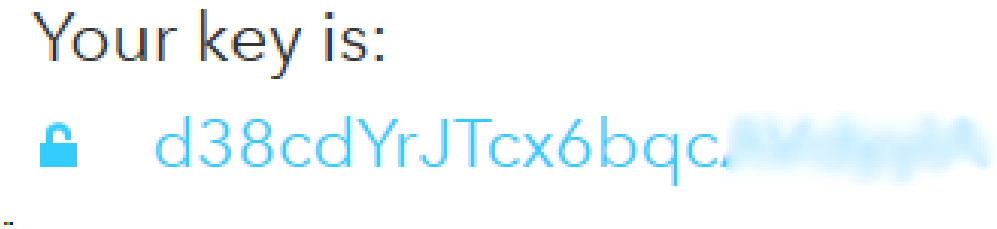

Figura 4.2: Password del canale MAKER

per ogni utente nella forma:

http://maker.ifttt.com/trigger/{event}/with/key/d38cdYrJTcx6bqc...

A questo punto si è pronti a creare una o più ricette che utilizzino il nostro canale maker come trigger.
Ad esempio:

Figura 4.3: Ricetta IFTTT che valuta la temperatura della stanza

Per cui bisogna inviare per far scattare l'evento maker chiamato "cold" una richiesta http all'URL:

http://maker.ifttt.com/trigger/cold/with/key/d38cdYrJTcx6bqc...

Oppure

Figura 4.4: Ricetta IFTTT che valuta la luce nella stanza

per cui bisogna inviare una richiesta http all'URL:

http://maker.ifttt.com/trigger/nolight/with/key/d38cdYrJTcx6bqc...

Il parametro event viene inserito al momento della creazione della ricetta ed è quello che viene utilizzato per creare parte dell'URL e passato nel momento dell'attivazione alla parte THAT, che la può utilizzare come variabile.
A questo punto dobbiamo utilizzare il MKR1000 per fare scattare i due trigger. Ovviamente i due trigger non verranno attivati casualmente ma rispondendo alla lettura di valori dati a dei sensori collegati ai pin della scheda. In questa modalità il nostro processore programmabile verrà quindi utilizzato come sensore verso il mondo reale. Per la precisione un sensore verrà usato per captare la temperatura mentre l'altro per rilevare la luminosità della stanza. MKR1000 ha alcuni dei suoi pin che sono utilizzati solo come input analogico, quindi possono leggere i valori di tensione che arrivano in quel pin.

Quindi per comporre il circuito avrò bisogno principalmente di due componenti:
- un fotoresistore e un sensore di temperatura.
- Il fotoresistore varia la sua resistenza interna in base alla luce che colpisce la propria superficie mentre il sensore di temperatura varia la tensione in base alla temperatura percepita

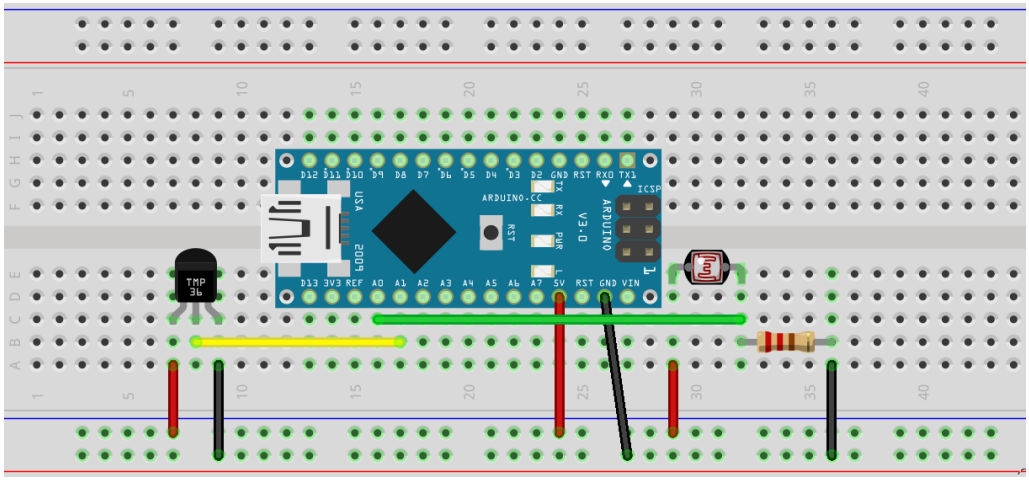

Figura 4.5: Circuito con MKR 1000, con un fotoresistore e un sensore di temperatura

Come si può vedere il fotoresistore è collegato al pin di input analogico A0 mentre il sensore di temperatura è collegato al pin di input analogico A1.
Analizziamo le parti più importanti del codice caricato nel MKR1000:

```
const int lightPin = A0;
const int temperaturePin = A1;

char hostname[] = "maker.ifttt.com";
char cold[]= "/trigger/cold/with/key/d38cdYrJTcx6bqc.......";
char nolight[] = "/trigger/nolight/with/key/d38cdYrJTcx6bqc.......";
```

Per prima cosa creo delle costanti, due per i pin in input, mentre le altre sono gli URL che dovranno essere chiamati in base alla lettura dei valori in in ingresso.

```
// Tenta la connessione Wifi:
  while ( status != WL_CONNECTED) {
    Serial.print("Provando la connessione con ");
    Serial.println(ssid);
    // Apre la connessione alla rete
    status = WiFi.begin(ssid, pass);

    // Attendiamo 10 secondi per dar tempo alla connessione di avvenire
    delay(10000);
  }
```

Nella fase di setup, cioè una porzione di codice che viene eseguita solo all'avvio della scheda, l'unica cosa degna di nota è la connessione. Fino a quando non si ottiene una connessione Wi-Fi, il codice non prosegue. I dati per connettersi sono stati precedentemente inseriti come costanti.

```
int lightVal=analogRead(lightPin);
int temperatureRead=analogRead(temperaturePin);
float temperatureVoltage = (temperatureRead/1024.0) * 5.0;
float temperatureVal = (voltage - 0.5) * 100;

if (lightVal <= 300 ) {
  Serial.println("\n\‘e buio.Manda la richiesta al server");
  // prova a connetterti
  if (client.connect(hostname, 443)) {
    Serial.println("connesso al server");
    // fai un request HTTP:
    client.print("POST ");
    client.print(nolight);
    client.println(" HTTP/1.1");
    client.print("Host: ");
    client.println(hostname);
    client.println("Connection: close");
    client.println();
```

```
    while(client.connected()){
      if(client.available()){
      String line = client.readStringUntil('\r');
        Serial.print(line);
      } else {
        // Non ho ricevuto risposta, attendere ancora
        delay(50);
      }
    }
    // Fatto
    Serial.println();
    Serial.println("Connessione chiusa");
    client.stop();
  }
  delay(10000);
}


if (temperatureVal < 18) {
  Serial.println("\fa freddo.Manda la richiesta al server");
  // prova a connetterti
  if (client.connect(hostname, 443)) {
    Serial.println("connesso al server");
    // fai un request HTTP:
    client.print("POST ");
    client.print(cold);
    client.println(" HTTP/1.1");
    client.print("Host: ");
    client.println(hostname);
    client.println("Connection: close");
    client.println();
    while(client.connected()) {
      if(client.available()) {
        String line = client.readStringUntil('\r');
        Serial.print(line);
      } else {
        // Non ho ricevuto risposta, attendere ancora
```

```
        delay(50);
      }
    }
  // Fatto
  Serial.println();
  Serial.println("Connessione chiusa");

  client.stop();
  }
delay(10000);
}
```

Una volta eseguito il setup si entra nella parte del codice chiamata loop, che viene richiamata all'infinito fino a quando non si toglie corrente alla scheda. Qui dentro non si fa altro che controllare lo stato dei due sensori. Se uno dei due ha un valore ritenuto arbitrariamente basso, quindi se l'ambiente risulta poco caldo o poco illuminato, si comincia una connessione con il server IFTTT. Questa connessione si fa tramite la porta 443 perchè proviamo a mandare un request con protocollo https, più sicuro dell http grazie alla criptazione dei dati inviati. Sulla base di quale dei due sensori ha generato l'evento si farà una richiesta ad un URL specifico. Una volta ricevuta dal server la risposta di corretta ricezione della richiesta viene chiusa la connessione e, a questo punto, la ricetta di IFTTT viene innescata.

Negli esempi sopracitati, ovviamente, non si è dato troppa importanza alla parte dell'azione compiuta dallo THAT, che come sempre sarebbe potuta essere una qualunque scelta tra i vari canali collegati a IFTTT.

4.4 Esempio that

Possiamo anche creare ricette IFTTT che utilizzino il canare maker come action, quindi il lato THAT. In parole povere significa che quando succede qualcosa in uno qualunque dei canali collegati si può far generare una richiesta http a IFTTT (nel senso che il canale maker genera una richiesta http). Nel momento in cui si crea la ricetta si può scegliere a chi fare la richiesta e in che formato. Quindi si deve predisporre qualcosa dall'altra

parte pronto a ricevere la richiesta, reagendo ad essa. Scegliamo di utilizzare nuovamente MKR1000, che questa volta deve avere la funzionalità di web server. La ricetta sarà la seguente:

Figura 4.6: Ricetta di IFTTT che ,tramite la connessione del prioprio smartphone al Wi-fi, apre la porta

Significa che quando il mio smartphone si collegherà alla rete Wi-Fi di casa, scatterà la ricetta e partirà una richiesta http all'indirizzo ip pubblico. In attesa dall'altra parte ci sarà il mio web server fatto con MKR1000 che intercetterà la richiesta. Nel momento in cui il web server locale riceverà una richiesta, sempre sul MKR1000 si attiverà un pin che permetterà al relay che controlla la mia porta di ingresso di aprirsi. In questo caso il nostro processore programmabile sarà un attuatore. Il circuito lo predispongo in modo da poter pilotare il relay con un cambio di tensione da basso ad alto in un pin ed inserendo ai capi del mio relay i cavi che permetteranno di aprire la porta. Senza entrare troppo nel dettaglio del circuito, ho bisogno di un transistor per poter pilotare con una tensione bassa, quella in uscita dai pin del MKR1000, una tensione più alta, quella necessaria per far funzionare il nostro relay collegato all' elettroserratura. In pratica, a livello di codice, bisogna solamente dare una tensione alta al pin D2 quando il mio web server riceve una richiesta effettuata da IFTTT. Il diodo serve solamente per proteggere il circuito e la scheda da inversioni e sbalzi di corrente che a volte possono capitare durante la chiusura del relay.

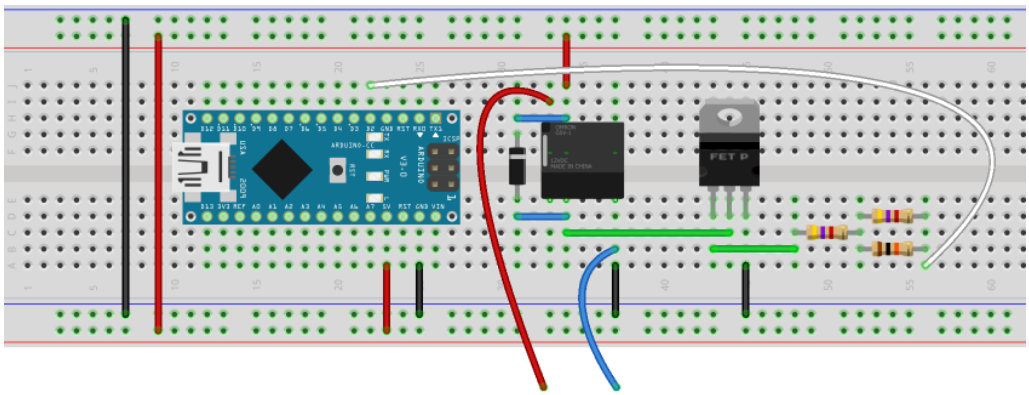

Figura 4.7: Circuito con MKR 1000, con relay collegato all'apertura della porta

Analizziamo ora le parti più importanti del codice:

```
const int relayPin = 2;

IPAddress Address(192, 168, 1, 100);
IPAddress Network(255, 255, 255, 0);
IPAddress Gateway(192, 168, 1, 1);
IPAddress Dns(8, 8, 8, 8);

WiFiServer server(80);
```

Prima di tutto creiamo una costante per il nostro pin che dovrà controllare l'elettroserratura. Poi assegnamo a 4 variabili di tipo IPAddress i dati necessari dell'indirizzo ip che il nostro server avrà. Come per ogni server è indispensabile avere un ip statico, quindi non assegnato tramite DHCP, in modo da poter indirizzare, tramite la programmazione del port forwarding del router, l'indirizzo pubblico al corretto indirizzo locale. Infine dichiariamo l'oggetto server, passandogli come parametro la porta su cui dovrà rimanere in ascolto. MKR1000 possiede un processore troppo poco potente per poter gestire la connessione criptata necessaria per utilizzare il protocollo https. Per questo utilizziamo la porta 80, quella standard del protocollo http.

```
// Tenta la connessione Wifi:
while ( status != WL_CONNECTED) {
  Serial.print("Provando la connessione con ");
  Serial.println(ssid);
  // Apre la connessione alla rete
  WiFi.config(Address, Dns, Gateway, Network);
  status = WiFi.begin(ssid, pass);

  // Attendiamo 10 secondi per dar tempo alla connessione di avvenire
  delay(10000);
}

server.begin();
```

Nella fase di setup, cioè quella eseguita una sola volta, mi connetto alla rete come nell'esempio precedente ma questa volta utilizzando l'indirizzo ip statico impostato prima. A connessione avvenuta il server verrà avviato.

```
WiFiClient client = server.available();
if (client) {
  Serial.println("nuovo client connesso");
  // una richiesta http finisce con una linea vuota quindi controllo
  // questa cosa
  boolean currentLineIsBlank = true;
  while (client.connected()) {
    if (client.available()) {
      char c = client.read();
      Serial.write(c);
      // se ottieni la fine di una riga (quindi hai ricevuto il
      // carattere \n) e la linea appena conclusa \‘e vuota
      // allora significa che la richiesta http verso il nostro
      // server \‘e finita e posso mandare una risposta
      if (c == '\n' && currentLineIsBlank) {
        // manda uno header standard
        client.println("HTTP/1.1 200 OK");
        client.println("Content-Type: text/html");
        client.println("Connection: close");
        // la connessione sar\‘a chiusa dopo il completamento
```

```
        // della risposta
        client.println();
        client.println("<!DOCTYPE HTML>");
        client.println("<html>");
        client.print("apriti sesamo! ");
        client.println("<br />");
  client.println("</html>");
  break;
}
if (c == '\n') {
        // sta cominciando una riga nuova
        currentLineIsBlank = true;
} else if (c != '\r') {
        // hai ricevuto un carattere su questa riga
        currentLineIsBlank = false;
}
    }
  }
  // dai tempo al web browser di ricevere la mia risposta
  delay(1);

  // chiudo la connessione
  client.stop();
  Serial.println("client disconnesso");


  // attivo la mia elettroserratura
  digitalWrite(relayPin,HIGH);
  delay(1000);
  digitalWrite(relayPin,LOW);
}
```

Nella parte del codice contenuta nel Loop di base si rimane in attesa di una connessione, catturando l'eventuale richiesta. Se una richiesta arriva, si aspetta di leggere ciò che il client ci manda, in attesa di non avere più nulla da ricevere. Quando questo succede rispondo al client e chiudo la connessione. Poi prima di ricominciare l'attesa mando un impulso al pin dell'elettroserratura per un secondo, aprendo così la porta.

4.5 Esempio trasmissione-ricezione

E' possibile utilizzare MKR1000 sia come parte THIS che parte THAT contemporaneamente. Ovviamente le due ricette di IFTTT dovranno essere separate. Molto probabilmente non sarà mai particolarmente utile avere entrambe le parti collegate, anche perchè lo sarebbero già trovandosi fisicamente sullo stesso microprocessore senza aver bisogno di passare per IFTTT, ma ad ogni modo è fattibile. L'unico dettaglio degno di nota riguarda la limitazione tecnica del microprocessore.

MKR1000 non dispone di un multithreading nativo, che significa che mentre il microprocessore è impegnato a fare qualcosa, questa è l'unica cosa che farà. Proprio per questo si deve implementare un semplice algoritmo di gestione dei tempi. Normalmente il nostro MKR1000 è in ascolto, in attesa della connessione di un client. Se arriva una richiesta da un client, si risponderà e si attiveranno i pin necessari e poi si chiuderà la connessione. Durante questo lasso di tempo i sensori verranno ignorati. Se i nostri sensori variano fino ad arrivare alla soglia, si fa l'invio della richiesta http. Fino a quando la richiestà non sarà stata completata, le richieste di eventuali client verranno ignorate. Questo piccolo problema risulta il più delle volte ininfluente, potrebbe però non essere più una soluzione soddisfacente se si devono monitorare valori con picchi istantanei e di breve durata.

Conclusioni

Internet of Things è destinato ad avere una crescita esponenziale nei prossimi anni fino a diventare una realtà pervasiva ed omnipresente nella quotidianità di chiunque. La mole di dati che verranno generati da questi nuovi dispositivi sarà immensa, eterogenea ma spesso incompleta. Incompleta perchè questi dati saranno solo una parte del processo decisionale di aziende, privati cittadini o città intere. Quindi una integrazione che permetta di utilizzarli al meglio diventerà una criticità entro breve.
Nella tesi viene quindi presentata una analisi formale ed esplicita delle specifiche necessarie per l'attuazione di questa visione futura tramite iPaaS, elencando e motivando le caratteristiche che deve possedere per poter essere considerato come uno dei pilastri delle tecnologie abilitanti dello IoT.
Con la parte pratica della tesi si è dimostrato come si possa rendere, sempre grazie all'utilizzo di un iPaaS e di un piccolo processore programmabile, qualunque oggetto una parte utilizzabile ed integrata di IoT, interfacciabile in maniera totalmente trasparente con qualunque altro servizio.
Ovviamente l'integrazione non è l'unico problema che ci troveremo ad affrontare per avere una reale attuazione del paradigma IoT. Ma avere dati ricavati dal mondo reale che possono essere applicati a qualunque catena decisionale o interagire sul mondo reale tramite dati ottenuti da fonti disparate è sicuramente un passo enorme nella giusta direzione per poter finalmente adattare tutte le incredibili potenzialità date dal big data non più solo al mondo virtuale ma anche al mondo reale, potendo così prendere decisioni che riguardano la nostra vita quotidiana con una visione omnicomprensiva, oculata e lungimirante, molto più di quanto faremmo noi come semplici esseri umani.

Ringraziamenti

Ringrazio tutti gli amici, le amiche e i parenti che mi hanno voluto bene, aiutato nelle piccole e nelle grandi cose, sostenuto negli strani momenti della vita e gioito assieme a me delle facezie come dell'indispensabile, è grazie a tutti voi se oggi sono arrivato fino a qui. Scusate se non faccio una lunghissima lista e non vi cito tutti quanti ma evito solamente la figuraccia, tipicamente mia, di dimenticarmi qualcuno.
Un grazie enorme e speciale a Vittoria, che mi ha fatto fretta perchè potessi essere tutto per lei quando arriverà.

Bibliografia

[1] Machina research. http://www.machinaresearch.com/.

[2] The internet of things. ITU Internet Reports, 2005.

[3] A. Al-Fuqaha et al. Internet of things: A survey on enabling technologies, protocols, and applications.

[4] K. Ashton. That "internet of things" thing. RFiD Journal.

[5] J. Baliga, R. Ayre, W. V. Sorin, K. Hinton, and R. S. Tucker. Energy consumption in access networks. San Diego, CA, February 2008. Proceedings of National Optical Fiber Communication Conference.

[6] M. Beigl, H. Gellersen, and A. Schmidt. Mediacups: Experience with design and use of computer-augmented everyday objects. Computer Networks, vol. 35, no. 4, 2001, pp. 401-409.

[7] J. Calbimonte, S. Sarni, J. Eberle, and K. Aberer. Xgsn: an open-source semantic sensing middleware for the web of things. Riva del Garda, Trentino, Italy, October 2014. 7th international workshop on semantic sensor networks, no. EPFL-CONF-200926.

[8] CASAGRAS. Casagras and the internet of things: Definition and vision statement agreed. 2009.

[9] CERP-IoT. Internet of things: Strategic research roadmap. 2009.

[10] A. Chao and F. Huaming. Internet of things design based on rfid technology in modern factory. Electronic Component & Device Applications, 2007,9 (12):75-77.

[11] I. Chlamtac, M. Conti, and J.-N. Liu. Mobile ad hoc networking:imperatives and challenges. Ad Hoc Netw. 1 (1) (2003) 13-64.

[12] X. G. Condori. Community cloud computing. 2013. Revista de Informacion Teconologia Y Sociedad, 70-72.

[13] EPoSS. Internet of things in 2020: Roadmap for the future. 2008.

[14] D. Evans. Internet of things - everything will change with the next internet wave. Cisco IBSG.

[15] Gartner. Who's who in middleware. 2004. Gartner Report.

[16] Gartner. Gartner it glossary - integration platform as a service (ipaas). Nov 2014.

[17] M. Gigli and S. Koo. Internet of things: Services and applications categorization. Jul. 2011. Adv. Internet Things, vol. 1, no. 2, pp. 27-31.

[18] D. Giusto, A. Iera, G. Morabito, and L. Atzori. The internet of things. Springer, 2009. ISBN: 978-1-4419-1673-0.

[19] R. Grewal and P. Pateriya. A rule-based approach for effective resource provisioning in hybrid cloud environment. Springer Berlin Heidelberg, 2013. New Paradigms in Internet Computing (pp. 41-57).

[20] I.-T. S. Group. New itu standards define the internet of things and provide the blueprints for its development. ITU, 2012.

[21] IDC. Idc smart cities index and its application in spain. 2011.

[22] V. Jacobson, D. Smetters, J. Thornton, M. Plasee, N. Briggs, and R. Braynard. Networking named content. Proceedings of ACM CoNEXT, Rome, Italy, 2009, pp. 1-12.

[23] W. Jansen and T. Grance. Guidelines on security and privacy in public cloud computing. February 2011. NIST special publication 800-144.

[24] G. Kortuem, F. Kawsar, D. Fitton, and V. Sundramoorthy. Smart objects as building blocks for the internet of things. Jan./Feb. 2010. IEEE Internet Comput., vol. 14, no. 1, pp. 44-51.

[25] M. T. Lazarescu. Design of a wsn platform for long-term environmental monitoring for iotapplications. IEEE Journal on Emerging and Selected Topics in Circuits and Systems (Volume: 3, Issue: 1, March 2013, pp. 45 - 54.

[26] G. Lewis. Basics about cloud computing. Software Engineering Institute Carniege Mellon University, Pittsburgh, 2010.

[27] T. S. Lø'pez, D. C. Ranasinghe, M. Harrison, and D. McFarlane. Using smart objects to build the internet of things. University of Cambidge.

[28] J. Manyika et al. Disruptive technologies: Advances that will transform life, business, and the global economy. San Francisco, CA, USA, 2013. McKinsey Global Instit. ISBN: 978-1-4419-1673-0.

[29] A. Marinos and G. Briscoe. Community cloud computing. Springer Berlin Heidelberg, 2009. Cloud Computing (pp. 472-484).

[30] F. Mattern. From smart devices to smart everyday objects. Springer, 2003. Smart Objects Conf. (SOC 03).

[31] P. Mell and T. Grance. The nist definition of cloud computing (technical report). National Institute of Standards and Technology: U.S. Department of Commerce, September 2011. doi:10.6028/NIST.SP.800-145. Special publication.

[32] MuleSoft. What is ipaas? gartner provides a reference model. 2012.

[33] G. Mulligan and D. Graýanin. A comparison of soap and rest implementations of a service based interaction independence middleware framework. 2009. Proceedings of the 2009 Winter Simulation Conference (WSC),Pages: 1423 - 1432/WSC.2009.5429290.

[34] D. Niyato, E. Hossain, and S. Camorlinga. Remote patient monitoring service using heterogeneous wireless access networks: architecture and optimization. IEEE Journal on Selected Areas in Communications 27 (4) (2009) 412-423.

[35] P.Harropand and R.Das. Wireless sensor networks 2010-2020. Cambridge, U.K., 2010. ID Tech Ex.

[36] M. Presser and A. Gluhak. The internet of things: Connecting the real world with the digital world. EURESCOM mess@ge - The Magazine for Telecom Insiders, vol. 2, 2009.

[37] H. Research. Machine-to-machine (m2m) and smart systems forecast,2010-2014. 2010.

[38] SAP. Sap: Internet of things: An integral part of the future internet. 2009.

[39] W. Shihuan. Applications analysis of iot based on rfid and wlan technologies. Information and Electronic Engineering, 2010,10,8 (5): 604-606.

[40] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. Souza, and V. Trifa. Soa-based integration of the internet of things in enterprise services. Los Angeles, Ca, USA, July 2009. IEEE ICWS 2009.

[41] L. Srivastava. Pervasive, ambient, ubiquitous: the magic of radio. Bruxelles, Belgium, March 2006. European Commission Conference "From RFID to the Internet of Things".

[42] N. Streitz et al. Designing smart artifacts for smart environments. Computer, vol. 38, no. 3, 2005, pp. 41-49.

[43] I. Toma, E. Simperl, and G. Hench. A joint roadmap for semantic technologies and the internet of things. Crete, Greece, June 2009. Proceedings of the Third STI Roadmapping Workshop.

[44] A. Vilamovska, E. Hattziandreu, R. Schindler, C. V. Oranje, H. D. Vries, and J. Krapelse. Rfid application in healthcare - scoping and identifying areas for rfid deployment in healthcare delivery. RAND Europe, February 2009.

[45] D. Washburn and U. Sindhu. Helping cios understand 'smart city' initiatives. Forrester, 2010.

[46] X. Xiaojiang, W. Jianli, and L. Mingdong. Services and key technologies of the internet of things. ZTE Commun., Shenzhen, China, vol. 2, p. 011, 2010.

[47] H. Zhou. Smarter earth: Deciphering internet of things. Publishing House of Electronics Industry, 2010.

[48] Z. Zhou, H. Zhang, X. Du, P. Li, and X. Yu. Prometheus: Privacy-aware data retrieval on hybrid cloud. April 2013. Proceedings of IEEE INFOCOM, (pp. 2643-2651).

[49] M. Zorzi, A. Gluhak, S. Lange, and A. Bassi. From today's intranet of things to a future internet of things: a wireless-and mobility-related view. 2010. IEEE Wirel Commun 2010;17(6):44-51.