Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

WoT Store: una piattaforma per l'interoperabilità semantica in contesti IoT e WoT

Relatore:
Chiar.mo Prof.
Marco Di Felice

Correlatore:
Dott.
Luca Sciullo

Presentata da:
Lorenzo Gigli

Sessione III
Anno Accademico 2017/2018

Sommario

Partendo da un'analisi dell'Internet of Things (IoT) e del Web of Things (WoT), delle loro criticità e delle possibili soluzioni, viene proposta la realizzazione di una piattaforma: WoT Store che, rispettando gli standard del W3C WoT, permetta una semplice e ampia operabilità su applicazioni e Thing su base semantica. Per quanto riguarda le applicazioni il WoT Store ne effettua la distribuzione, la ricerca e l'installazione. Permette inoltre la ricerca e la visualizzazione interattiva delle Thing disponibili e la selezione delle applicazioni compatibili con esse. Il lavoro è da considerarsi in una fase dinamica, non solo in quanto necessita di continuo adeguamento con gli standard W3C, ma anche perchè si può prevedere l'implementazione della compra-vendita delle applicazioni e dei dati delle Thing tramite criptovaluta.

Indice

Elenco delle figure

Elenco listati di codice

Introduzione

Per quanto il concetto di Internet of Things (IoT) non sia nato recentemente, nell'ultimo decennio si è assistito via via a un'espansione vertiginosa del settore che presumibilmente non conoscerà battute d'arresto negli anni a venire. Infatti mentre nel 2015 il numero di dispositivi connessi era di 15.41 miliardi, ad oggi la cifra è di 26.66 e la stima per il 2025 raggiunge i 75.44, generando un mercato che attualmente è valutato intorno ai 1710 miliardi, in continua crescita[Sta19a][Sta19b].

La principale criticità dell'IoT è costituita dalla difficile/impossibile interoperabilità tra i numerosissimi prodotti esistenti, che parlano protocolli e formati diversi non in grado di dialogare tra loro.

Al momento attuale si è cercato di affrontare questo problema tramite il Web of Things (WoT), utilizzando quindi gli standard e le tecnologie web già ampiamente diffusi, consolidati e adatti a costruire applicazioni fortemente scalabili. Il World Wide Web Consortium (W3C), sostenendo l'idea del WoT, nel 2015 ha cominciato a lavorare alla sua standardizzazione, con lo scopo di rimediare alla frammentazione dell'IoT grazie a una serie di componenti complementari che permettano l'integrazione tra differenti piattaforme e domini applicativi. Tale processo dovrebbe sfociare in una serie di W3C Recommendation entro giugno 2019. Oltre ai membri del W3C partecipano al lavoro collaboratori di un grande numero delle più grosse aziende tecnologiche a livello mondiale, fortemente motivate al perseguimento di questa operazione.

Lo scopo del lavoro oggetto di questa tesi è la progettazione e l'imple-

mentazione di una piattaforma (WoT Store) che gestisca applicazioni e thing aderendo con precisione all'architettura proposta dal W3C. La piattaforma è caratterizzata da un'interfaccia di ridotta complessità, che sottende un sistema fortemente automatizzato in grado di semplificare e velocizzare l'attività dell'utente. Una peculiarità del WoT Store è quella di essere fruibile anche da sviluppatori non necessariamente specializzati in questo ambito, purchè in possesso di una competenza tecnica medio-elevata.

Se, per quanto riguarda le funzionalità di base, esiste un parallelismo tra il WoT Store e i market di applicazioni utilizzati in ambiente mobile, tuttavia il WoT Store introduce alcune funzionalità uniche, tra cui appaiono particolarmente interessanti l'approccio semantico, la possibilità di installare contemporaneamente le applicazioni su diversi dispositivi, la renderizzazione delle thing.

In questo elaborato sono esposte le varie tappe di studio e di lavoro svolto per la realizzazione di questo sistema.

In una prima parte (capitoli 2 e 3), viene presentato lo stato dell'arte. Il capitolo 2 contiene una panoramica sull'IoT, con alcuni cenni storici, la sua evoluzione, le sue caratteristiche, le sue criticità. Quindi prende in esame il WoT, illustrandone l'utilità e l'utilizzo e infine introduce il lavoro del W3C (tutt'ora in corso) esaminandone i documenti prodotti. Nel capitolo 3 viene analizzata l'architettura del W3C WoT, mostrandone i casi d'uso e gli elementi principali e l'implementazione del Servient.

La seconda parte dell'elaborato (capitoli 4 e 5) riguarda in maniera specifica il sistema sviluppato. Nel capitolo 4 viene presentata l'idea iniziale che ha dato origine a questo progetto, i requisiti funzionali e tecnici, l'architettura. Nel capitolo 5 sono esposte nel dettaglio le tecnologie adottate e i moduli software realizzati. Il capitolo 6 riporta la sperimentazione eseguita per la validazione del lavoro.

Parte I

Stato dell'Arte

Capitolo 1

Panoramica generale

1.1 Internet of Things

Internet of Things (IoT) è un neologismo riferito all'estensione di Internet al mondo degli oggetti e dei luoghi concreti. Questo termine viene utilizzato per la prima volta nel 1999 da Kevin Ashton, ricercatore presso il MIT, Massachussets Institute of Technology, il quale, all'epoca, si riferiva solo ed esclusivamente a quei dispositivi connessi che potevano essere identificati in modo univoco tramite la tecnologia Radio-Frequency IDentification (RFID).

Si può immediatamente notare come, nonostante se ne sia cominciato a parlare solo recentemente (negli ultimi anni), in realtà è un concetto che ha origini remote e che durante un lungo processo di raffinazione e espansione è arrivato alla complessità e vastità che conosciamo oggi.

Proprio per questo motivo esprimere la vera essenza dell'IoT in una definizione è praticamente impossibile, tuttavia possiamo affermare che l'Internet moderno si spinge oltre una semplice collezione di contenuti multimediali: si dirama nel mondo fisico, nel mondo real-time utilizzando tutta una serie di dispositivi interconnessi di varie dimensioni.

"A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on, existing and evolving, interoperable information and communication technologies"[Zav15].

L'evoluzione può essere suddivisa in diverse fasi come mostrato nella figura 1.1. L'IoT ha cominciato la sua ascesa con l'uso della tecnologia RFID, che viene sempre più utilizzata nella logistica, nella produzione farmaceutica, nella vendita al dettaglio, ecc. L'emergere delle tecnologie sensoristiche wireless ha permesso di estendere in modo significativo le capacità sensoriali dei dispositivi, potenziandone l'intelligenza spaziale/ambientale e il controllo autonomo. Ad oggi, sono coinvolte nell'IoT numerose tecnologie, come le reti di sensori wireless (WSNs), i codici a barre, sensori intelligenti (sensori che compiono una qualche azione predefinita quando ricevono il giusto input), RFID, NFC, comunicazioni wireless low-energy, cloud computing, e molte altre ancora[Sha15].

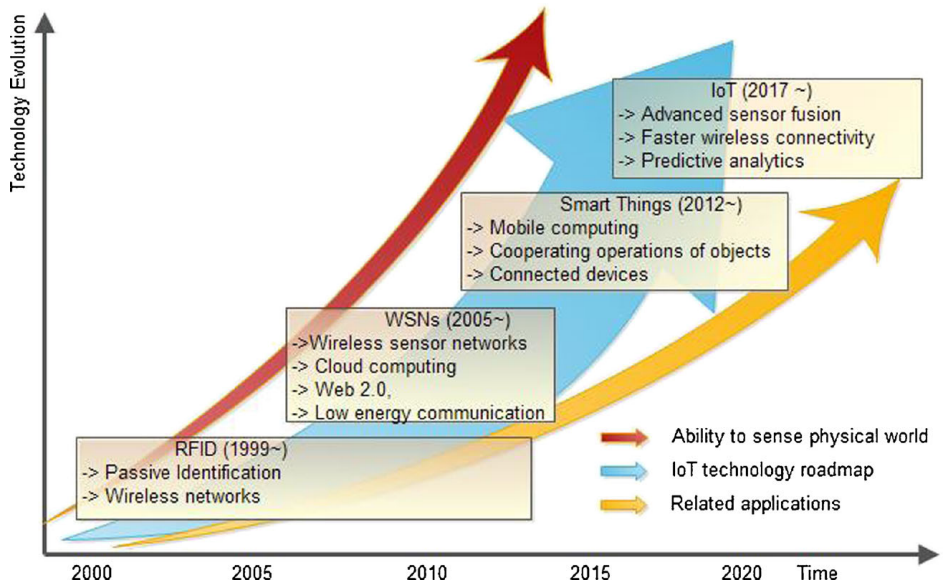

Figura 1.1: Evoluzione dell'IoT [1]

fonte: [Sha15]

La continua evoluzione di queste tecnologie e dei dispositivi embedded ha portato sul mercato una nuova tipologia di oggetti: *smart things*. Una *smart thing* (a cui ci riferiremo come Thing) non è altro che un oggetto fisico "potenziato" digitalmente tramite una o più delle seguenti caratteristiche:

- Sensori (temperatura, luce, movimento, ecc.)
- Attuatori (schermi, motori, ecc.)
- Computazione (la capacità di eseguire programmi e logica)
- Interfacce di comunicazione (cablate o wireless)

Queste Thing vanno ad estendere direttamente il mondo in cui viviamo aprendo le porte a nuovi tipi di applicazioni che le sfruttano singolarmente o in modo combinato. Col passare del tempo sempre più computer verranno integrati nell'ambiente in cui viviamo, rendendo il nostro grado di interazione col mondo fisico sempre più fluido ed effettivo, permettendoci di manipolare e ottenere informazioni da qualsiasi tipo di oggetto. Le Thing possono variare da un semplicissimo prodotto munito di Auto-ID tag (codici a barre, codici QR, NFC e RFID tag) per poterlo monitorare durante i suoi spostamenti (un oggetto comprato su Amazon), fino a sistemi ad alta complessità come ad esempio un'automobile (vedi figura 1.2). La caratteristica comune è che tutte quante, direttamente o indirettamente, possono essere raggiunte e utilizzate da altre applicazioni attraverso la rete Internet.

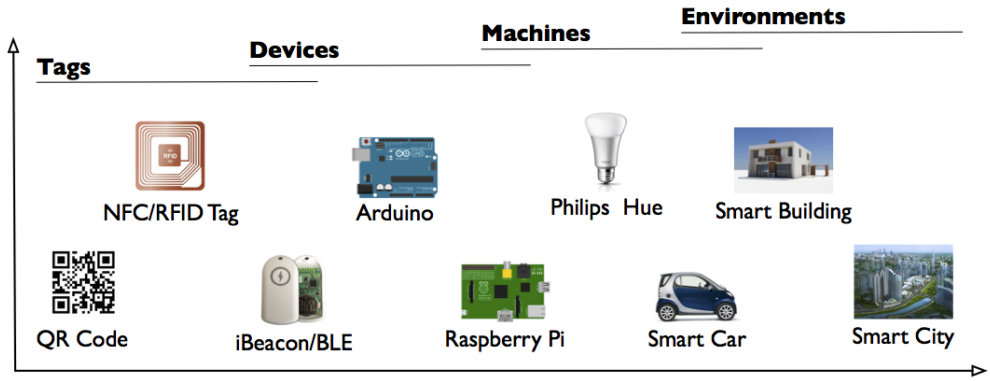

Figura 1.2: Alcuni esempi di Thing: dalle più semplici a quelle più complesse [2]
fonte: [Tri16]

Sfortunatamente la sua popolarità e diffusione ha portato in campo troppi giocatori. Le società tecnologiche stanno sviluppando soluzioni indipendentemente l'una dall'altra, utilizzando diverse piattaforme e framework e questo si traduce in molti dispositivi differenti che non possono in alcun modo integrarsi tra loro[Eas17].

Il dominio IoT è suddiviso tra vari protocolli di rete low-energy (ZigBee, ZWave e Bluetooth), protocolli di rete tradizionali (Ethernet, Wi-Fi) e persino connessioni cablate. Questi protocolli sono progettati per applicazioni specifiche con delle caratteristiche distintive. Come è facile intuire, la risoluzione del problema di interoperabilità a questo livello richiederebbe la standardizzazione a livello hardware[Pra15][Som13].

Per far sì che l'IoT diventi qualcosa di reale, abbiamo bisogno di un nuovo layer per permettere ai dispositivi e alle applicazioni di comunicare tra loro, a prescindere da come sono connesse fisicamente. Il concetto che sta dietro al Web of Things (WoT) è proprio questo: invece di andare a costruire un nuovo protocollo da zero, possiamo utilizzare i protocolli, gli standard e i blueprint del web che sono già ampiamente diffusi, rodati e nati per costruire applicazioni fortemente scalabili. Inoltre portando l'astrazione a livello applicativo è possibile coinvolgere una cerchia di sviluppatori sicuramente più ampia rispetto a quelli che lavorano esclusivamente a basso livello.

1.2 Web of Things

Come descritto nella sezione 1.1, le limitazioni dell'IoT diventano immediatamente chiare nel momento in cui si vogliano integrare diversi tipi di dispositivi, di aziende diverse, all'interno di un'unica applicazione o sistema. Il problema di eterogeneità è di natura più commerciale che tecnica: nel corso degli anni sono stati proposti tantissimi standard da diversi organismi di standardizzazione, alleanze industriali e venditori in generale. Purtroppo nessuno di questi è stato abbastanza forte da trascinare gli altri in un'unica direzione permettendo di creare un "protocollo universale". Questa situazione ha creato anche un forte disagio per l'utente finale che si vede molto spesso costretto a scegliere tutti i prodotti di un unico produttore per avere una certa garanzia di interoperabilità. Tuttavia, anche in questo caso, le funzionalità e la qualità del software sono molto spesso limitate ai pochi utilizzi pensati e proposti dal produttore stesso.

Il WoT punta a virtualizzare il concetto di Thing, trattandole come proxie dei corrispettivi oggetti fisici o astratti, accessibili tramite le tecnologie web[Rag15]. Il WoT si basa esclusivamente sui protocolli e strumenti dello strato Application del modello OSI (figura 1.3).

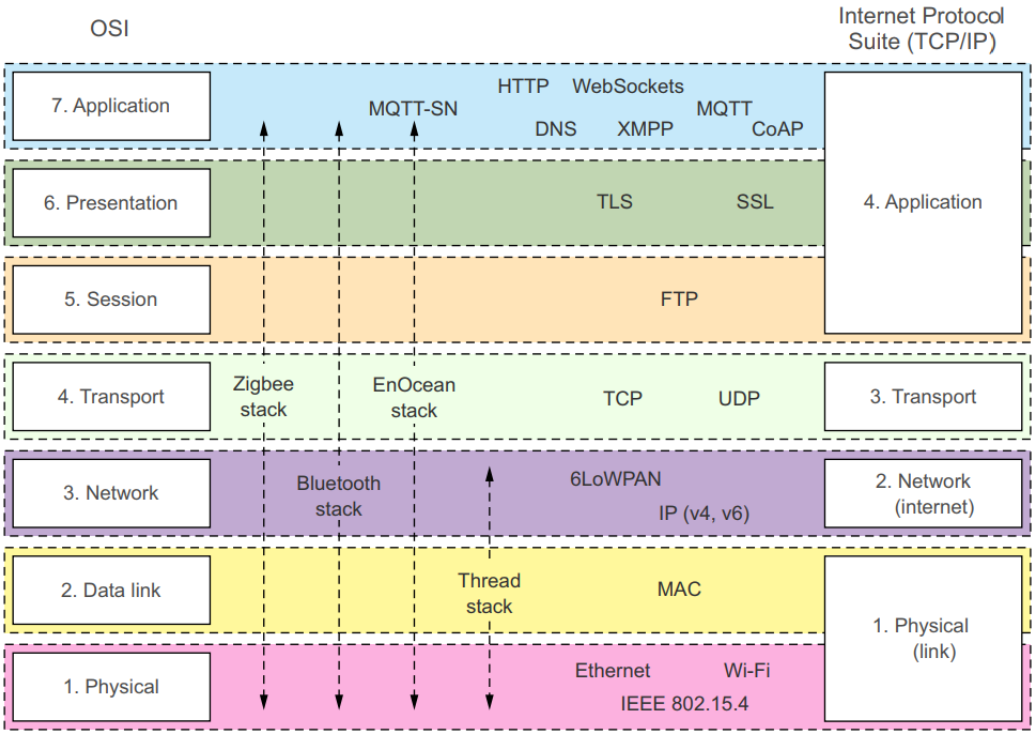

Figura 1.3: Modello OSI a confronto con Internet Protocol Suite (TCP/IP) [3]
fonte: [Tri16]

A differenza del modello OSI o IPS (figura 1.3), il modello architetturale del WoT non è composto da rigidi strati (layer), ma bensì da livelli che aggiungono funzionalità, come mostrato nella figura 1.4. Ogni livello permette alle Thing di raggiungere un grado di integrazione col web più elevato, rendendole via via sempre più accessibili alle applicazioni e agli utenti. Il modello del WoT è da posizionare esattamente alla fine dei modelli OSI o IPS, come una vera e propria espansione dello strato Application. Questo concetto fornisce una solida base di partenza, perché sta ad indicare che non c'è bisogno di "preoccuparsi" del funzionamento e dell'implementazione dei livelli sottostanti (1-6).

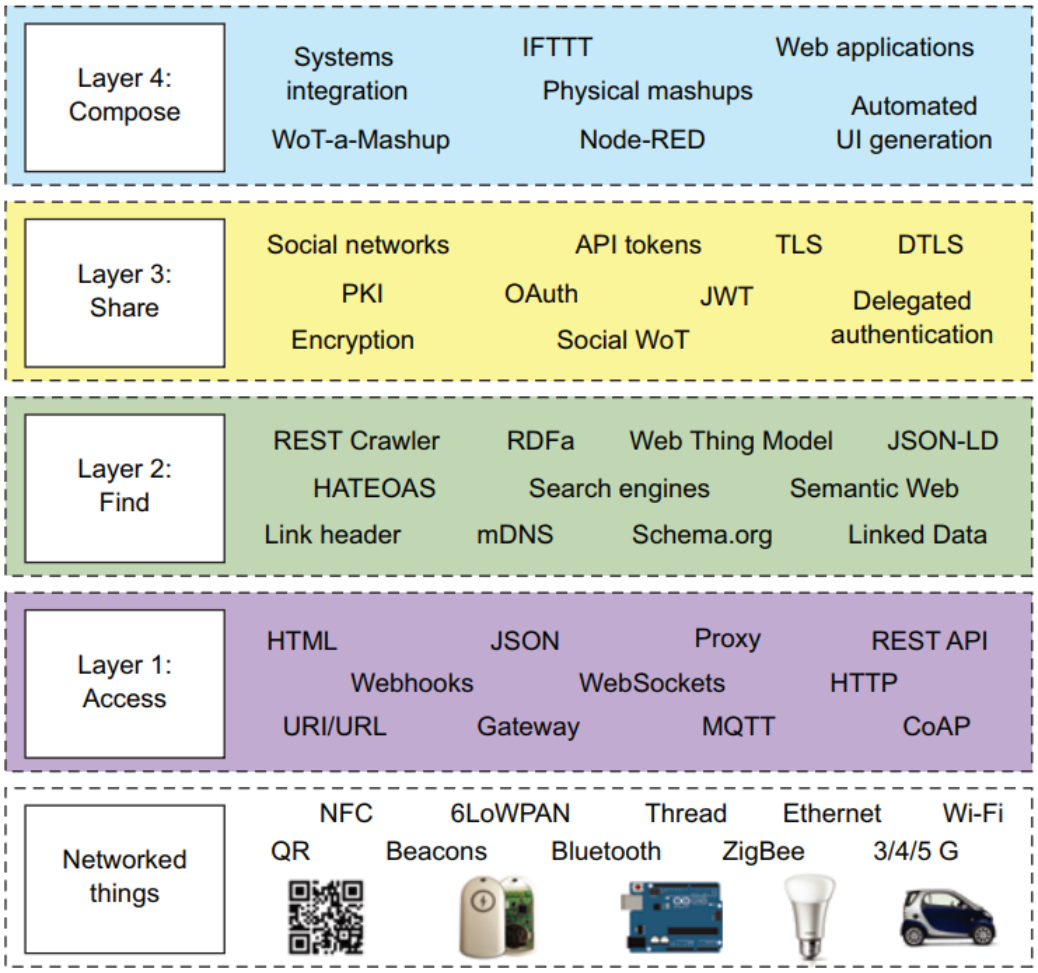

Figura 1.4: Lo stack architetturale del WoT [4]
fonte: [Tri16]

Andiamo ad analizzare i vari livelli del modello WoT descrivendone alcune tecnologie e il loro scopo:

- **Livello 1: Access**
  Questo livello si occupa dell'accesso delle Thing a Internet e garantisce che espongano i loro servizi tramite un'API web. Si tratta del livello principale, perché funge da vero e proprio punto d'accesso, trasformando le Thing in oggetti programmabili. Le Thing possono esporre i propri servizi in vari modi: tramite HTTP, utilizzando un'API REST-

ful e JSON come formato per i dati; tramite WebSocket o MQTT[5] se c'è bisogno di una comunicazione real-time / event-drive. Ovviamente non tutte le Thing sono capaci di parlare tramite protocolli web direttamente, ma ciò non impedisce che possano fare parte del WoT. Come vedremo nella sezione 2.1 esistono dei pattern specifici per gestire queste integrazioni.

- **Livello 2: Find**
  L'obiettivo di questo livello è fornire un modo per trovare e localizzare le Thing sul web, quindi è fortemente influenzato dal web semantico[Gui11]. L'approccio è quello di riutilizzare gli standard semantici del web per descrivere le Thing e il loro servizi. Come vedremo nella sezione 2.3.1, RDF[6] e JSON-LD[7] giocano sicuramente un ruolo fondamentale in questo contesto. Ciò consente la ricerca delle Thing attraverso i motori di ricerca, il facile sviluppo di applicazioni e l'interazione machine-to-machine basata su un piccolo insieme di formati e standard ben definiti.

- **Livello 3: Share**
  Il WoT si basa in gran parte sull'idea di Thing che "pubblica" dati sul web, dove possono essere applicate delle tecniche di big-data più sofisticate. La responsabilità del livello Share è proprio quella di definire come questi dati devono essere condivisi e sopratutto con quali misure di sicurezza. Per questo motivo qui vengono presi in considerazione i meccanismimi di autenticazione come API token, OAuth,[8] ecc. e integrati nelle API delle Thing.

- **Livello 4: Compose**
  Una volta giunti al 4° livello, abbiamo a disposizione una serie di Thing,

[5] http://mqtt.org/
[6] https://www.w3.org/RDF/
[7] https://json-ld.org/
[8] https://oauth.net/

che possono essere trovate facilmente sia in modo manuale, sia automatico e che possono essere condivise in modo sicuro con altri. Gli strumenti del livello Compose possono spaziare dai web toolkit (JavaScript SDK), una dashboard con dei widget programmabili, fino ad intere piattaforme di mashup come Node-RED.[9] Questi strumenti possono arrivare a consentire alle persone di creare applicazioni utilizzando i servizi WoT senza richiedere competenze di programmazione[Tri16].

Mentre è sempre possibile dare a Thing individuali presenza sul web, la tendenza è quella di aggregarle in gruppi più o meno numerosi attraverso piattaforme cloud. Queste piattaforme possono essere suddivise in cinque categorie:

- **Prodotti IoT "web-ready"**
  Si tratta di prodotti connessi a Internet che utilizzano servizi basati sul web per permettere l'interazione da parte dell'utente, attraverso applicazioni mobili e browser (es. termostato NEST[10]).

- **Piattaforme IoT web-centriche**
  Riguarda l'IoT industriale e i sistemi machine-to-machine (es. ThingWorx,[11] RealTime.io[12]). Il loro obiettivo è fornire strumenti per costruire soluzione end-to-end, facilitandone l'accesso per mezzo di tecnologie web basate su API proprietarie e i loro modelli di programmazione.

- **WoT Hubs**
  Piattaforme che aggregano collezioni di stream di dati per immagazzinare le informazioni dei sensori (es. ThingSpeak[13]). Includono applicazioni per l'elaborazione, la visualizzazione, l'integrazione e la condivisione dei dati, permettendo ai diversi utenti di trarne reciprocamente vantaggio.

[9] https://nodered.org/
[10] https://nest.com/
[11] https://thingworx.com/
[12] https://realtime.io/
[13] https://thingspeak.com/

- **Reti di sensori**
  Sono sistemi che utilizzano massivamente i protocolli web per creare vaste applicazioni di sensori.

- **Strumenti di mashup**
  Facilitano gli sviluppatori nel processo di creazione di applicazione che combinano dati e funzionalità attraverso diverse fonti dati sia fisiche che virtuali (es. IFTTT[14]).

Dati i diversi tipi di approccio utilizzati dai vari gruppi WoT, è evidente come si renda necessario superare i confini dei sistemi e delle piattaforme in modo tale da livellare le differenze. Tutto ciò impone di eseguire delle assunzioni sui vari tipi di Thing, in modo tale da compensare le differenti metodologie d'accesso, di sicurezza, di rappresentazione, ecc. Con l'espansione del WoT si è verificata una criticità, consistente nella creazione di "isole" che parlano linguaggi diversi: ciò rischia di rallentare l'ulteriore avanzamento del suo sviluppo[Mic13]. Il W3C ha perciò deciso di affrontare la problematica, per arrivare a un accordo sulle modalità con cui le Thing devono essere rappresentate, trovate, fruite.

1.3 W3C WoT

Il World Wide Web Consortium (W3C) è una comunità internazionale i cui membri, uno staff a tempo pieno e il pubblico, lavorano insieme per sviluppare gli standard web[19a].

Nel 2015 il W3C ha istituito un working group per lavorare alla standardizzazione del WoT[15]. Lo scopo del gruppo è quello di contrastare la frammentazione dell'IoT attraverso una serie di componenti standard complementari (Metadati, API, ecc.) che consentono una facile integrazione tra piattaforme IoT e domini applicativi[19c]. Il lavoro è stato portato avanti negli anni successivi e sono stati prodotti una serie di documenti: 3 normativi

[14] https://ifttt.com/

che devono essere seguiti alla lettera per soddisfare i requisiti dello standard e 3 informativi che invece sono pensati per aiutare il lettore a comprendere i concetti presentati negli elementi normativi[19b].

Documenti normativi:

- **WoT Architecture**
  Questa specifica definisce l'architettura di alto livello per i singoli blocchi del WoT e le necessarie configurazioni della piattaforma. Inoltre descrive i possibili scenari di deploy presi in considerazione dal WoT.

- **WoT Thing Description**
  Si tratta della parte più importante, perché definisce un'ontologia per la descrizione semantica delle Thing in termini di dati e modelli di interazione che espongono e che possono essere utilizzati dalle applicazioni. Inoltre descrive anche i metadati per quanto riguarda la sicurezza e le comunicazioni.

- **WoT Scripting APIs**
  Un'API per le applicazioni completamente indipendente dalla piattaforma, che serve a gestire l'interazione Thing-to-Thing e il loro lifecycle.

Documenti informativi:

- **WoT Binding Templates**
  Fornisce una serie di design pattern ed estensioni al vocabolario della WoT Thing Description che permettere l'adattamento ai formati, schemi di payload e tipi di dati dei vari protocolli e standard in uso per connettere le Thing tra loro.

- **WoT Security and Privacy Considerations**
  Definisce i requisiti di sicurezza generali per un sistema WoT utilizzando un "threat model".[15] Il WoT threat model chiarisce quali sono i

[15]https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling

principali soggetti interessati alla sicurezza, le risorse rilevanti, i possibili attaccanti, le superfici d'attacco e infine le minacce per un sistema WoT.

- **WoT Test Cases**
  Questo documento presenta i test corrispondenti alle questioni tecniche affrontate dal working group. Inoltre, aiuta a valutare l'interoperabilità tra le implementazioni della suite di test e le implementazioni esterne (es. progetti open source).

Ad eccezione del WoT Test Case, tutti questi documenti hanno visto la loro prima pubblicazione verso fine 2017 e diventeranno delle W3C Recommendation[16] a Giugno 2019[19b].

Ad oggi all'interno del gruppo sono presenti oltre 100 partecipanti tra i quali, oltre ai membri W3C, collaboratori attivi di aziende come Alibaba Group, Baidu Inc., Ericsson, Fujitsu Ltd., Hitachi Ltd., Huawei, Intel Corporation, Oracle Corporation, Panasonic Corporation, Siemens AG e altre[17].

[16] https://www.w3.org/standards/faq

Capitolo 2

W3C WoT Architecture

L'obiettivo dell'architettura WoT del W3C è quello di permettere l'interoperabilità tra diversi sistemi IoT, indipendentemente dal loro stack d'implementazione e dai protocolli di rete che usano. Le specifiche del W3C WoT sono state rilasciate verso la fine del 2017 e nelle sezioni successive farò riferimento all'ultimo "draft" disponibile: [W3C19a]. Bisogna considerare che i documenti sono in rapida evoluzione (ed in parte incompleti) e possono ancora subire dei cambiamenti importanti sia da un punto di vista strutturale, sia di contenuto.

2.1 Casi d'uso

I casi d'uso che vengono presi in esame del W3C WoT e che serviranno per andare a costruire l'architettura astratta discussa successivamente sono:

- Consumer
- Industriale
  - monitorare e prevedere possibili guasti e anomalie dei macchinari
  - tracciare le spedizioni, la loro condizione e qualità
  - leggere i contatori residenziali, commerciali e industriali

- Smart Buildings
  - monitorare il consumo energetico in tutto l'edificio
  - raccogliere dei feedback sulla soddisfazione degli inquilini
- Automobili
  - monitorare lo stato operativo e ottimizzare la manutenzione
  - migliorare la sicurezza del conducente grazie a notifiche puntuali sulle strade più pericolose e lo condizioni del traffico
- Agricoltura
  - monitorare le condizioni del terreno e creare dei piani ottimali per l'irrigazione, la concimazione
  - monitorare le condizioni di produzione
- Sanità
  - raccolta e analisi dei dati degli studi clinici
  - controllare i pazienti dopo l'ospedalizzazione
- Smart Cities
  - monitorare ponti, dighe, canali e altre strutture
- Monitoraggio ambientale
  - inquinamento dell'aria, dell'acqua, radioattività, ecc.

Sono definiti anche dei pattern per illustrare come i dispositivi interagiscono con i loro controller, altri dispositivi, agenti e server: possiamo trovare schemi semplici come Device Controllers e Thing-to-Thing, fino a delle casistiche più articolate tipo Digital Twin e Cross-domain Collaboration. Questi ultimi due casi sono sicuramente i più interessanti, infatti il primo è sostanzialmente una rappresentazione virtuale di uno o più dispositivi, mentre il secondo è una dimostrazione di come si possano connettere tra loro numerosi sistemi con pattern differenti in un unico ecosistema.

2.2 Architettura

2.2.1 Panoramica generale

Nella figura 2.1 possiamo vedere una panoramica di quella che potrebbe essere un'architettura WoT, mostrando tre livelli di applicabilità differenti:

- dispositivo
- gateway
- cloud

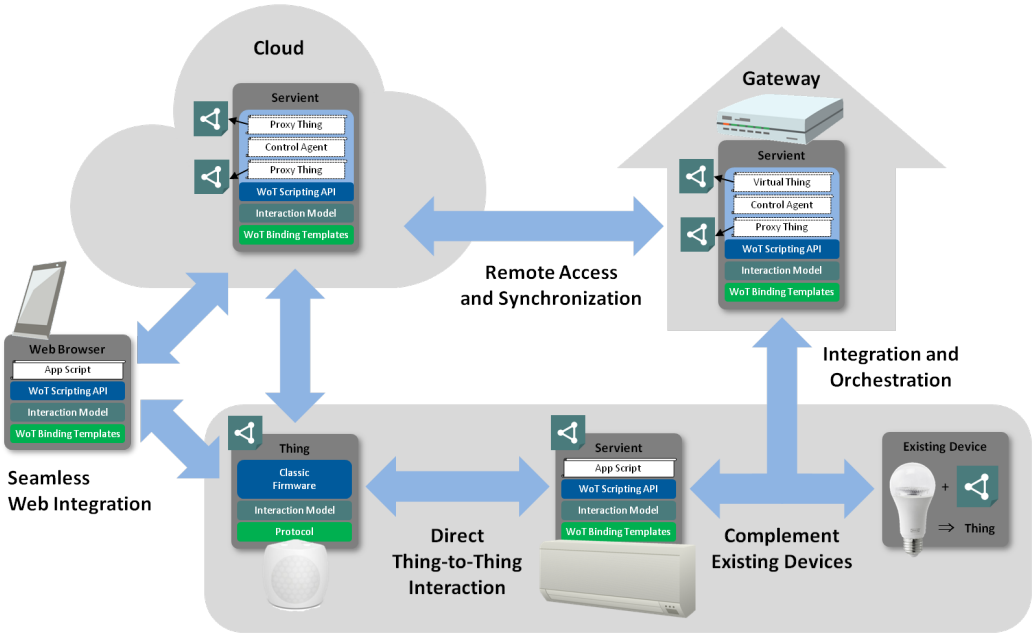

Figura 2.1: Architettura astratta del W3C WoT [1]
fonte: [W3C19a]

2.2.2 Architettura di alto livello

Partiamo mostrando una configurazione base formata da un dispositivo e un'applicazione.

Figura 2.2: Architettura di alto livello di un'applicazione e dispositivo [2]
fonte: [W3C19a]

Le funzioni di un dispositivo sono descritte da una Thing Description (TD). Una TD descrive, tra le altre cose, l'identificativo, le funzioni, gli attributi di un dispositivo e i suoi protocolli di comunicazione.

Ogni dispositivo appartenente al WoT deve avere una TD corrispettiva. Le applicazioni possono riconoscere un dispositivo ottenendo la sua TD. Idealmente, si può pensare che un dispositivo abbia un ExposedThing che fornisce un'interfaccia conforme al TD. D'altra parte, un'applicazione si può considerare come provvista di un ConsumedThing che fornisca una funzionalità d'interfaccia utilizzabile da un'applicazione. Un'applicazione, processando una TD, può generare un ConsumedThing. Le comunicazioni tra un'applicazione e un dispositivo sono realizzate da ConsumedThing e ExposedThing che si connettono tra loro e scambiano messaggi.

Andiamo ad esaminare il caso in cui un'applicazione e il dispositivo si connettano tramite un proxy.

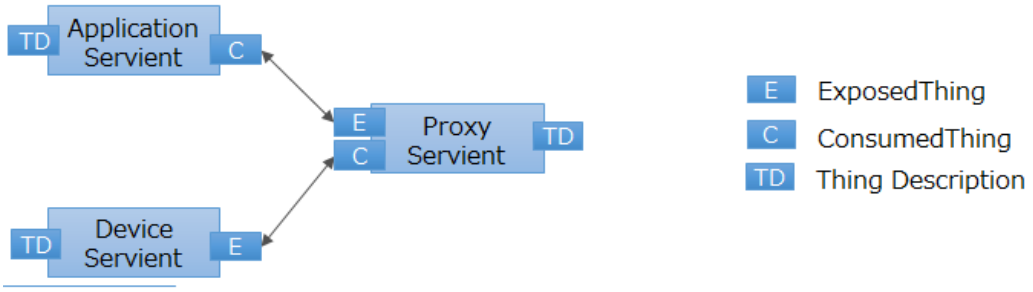

Figura 2.3: Architettura di alto livello con un proxy [3]
fonte: [W3C19a]

Un proxy contiene sia la funzionalità ExposedThing, sia ConsumedThing e ritrasmette i messaggi che vengono scambiati tra un'applicazione e un dispositivo. In un proxy la ConsumedThing genera un'astrazione del dispositivo, in questo modo l'applicazione può accedere al dispositivo-ombra attraverso l'ExposedThing del proxy.

L'ExposedThing e il ConsumedThing del proxy possono comunicare con protocolli differenti. Anche in presenza di più dispositivi che usano protocolli o metodi di autenticazione differenti, l'applicazione può comunicare con tutti attraverso il proxy in un'unica modalità.

Un proxy crea una ConsumedThing basata su una TD e genera un'altra TD per il dispositivo-shadow. La nuova TD per il dispositivo utilizza anche un nuovo identificativo rispetto a quello della TD originale e, se necessario, cambia anche i protocolli utilizzati. A questo punto il proxy crea un ExposedThing che funge da Thing per la nuova TD. Infine un'applicazione comunica con un dispositivo utilizzando il proxy, attraverso una ConsumedThing che funziona in base alla TD per il dispositivo-ombra.

2.2.3 Thing

Una Thing non è altro che un'astrazione di un'entità fisica o virtuale i cui metadati e interfacce sono descritti tramite una TD.

Una Thing può essere la rappresentazione digitale di un dispositivo IoT fisico, che ne fornisce anche un'interfaccia per l'interazione. Può trovarsi an-

che su un gateway o in cloud, diventando la ”porta d'accesso” per un'entità con cui altrimenti non sarebbe possibile comunicare direttamente (es. un dispositivo legacy o una stanza). Alcune entità astratte possono anche essere una composizione di Thing: una stanza che contiene numerosi sensori e attuatori, ognuno dei quali è a sua volta una Thing.

Tuttavia, ci possono essere delle Thing che non forniscono un'interfaccia d'interazione WoT e consistono solo di metadati rilevanti per l'applicazione.

2.3 Elementi principali

Andiamo ad esaminare quali sono i 3 elementi (Building Blocks) necessari per passare dall'architettura astratta alla sua implementazione.

2.3.1 WoT Thing Description

Il documento WoT Thing Description[W3C19d] definisce un modello di informazione basato su un vocabolario semantico, rappresentato in un formato basato su JSON. Sia il modello, che la sua rappresentazione sono allineati con i Linked Data[Ber06], in questo modo le varie implementazioni possono scegliere di utilizzare JSON-LD[Kel18] e i database a grafo per sfruttare al massimo l'elaborazione dei metadati. Per ora, JSON-LD offre un buon compromesso tra semantica comprensibile alla macchina e usabilità per gli sviluppatori.

Una TD descrive l'istanza di una Thing con una serie di metadati generici (*id*, *name*, *description*, ecc.): quelli relativi alle interazioni, al modello dati, comunicazione, link e meccanismi di sicurezza. La TD è costruita attorno a un modello di interazione formale in grado di supportare più paradigmi di messaggistica (request-response, publish-subscribe, ecc.). I pattern di interazione predefiniti sono: *Property*, *Action* e *Event*. Le proprietà astraggono quei dati che possono essere letti e spesso anche scritti. Le azioni rappresentano i processi invocabili che possono essere eseguiti per un certo periodo di tempo; tuttavia potrebbero anche essere classiche interazioni RPC. Gli eventi

descrivono le fonti che inviano messaggi ”push” asincroni. La TD può essere vista come l'index.html per le Thing, perché fornisce un punto d'ingresso per scoprire i servizi esposti e le varie risorse presenti.

```
{
  "@context": "http://www.w3.org/ns/td",
  "id": "urn:dev:wot:com:example:servient:lamp",
  "name": "MyLampThing",
  "security": [{"scheme": "basic", "in": "header"}],
  "properties": {
    "status": {
      "type": "string",
      "forms": [{
        "href": "https://mylamp.example.com/status"
      }]
    }
  },
  "actions": {
    "toggle": {
      "forms": [{
        "href": "https://mylamp.example.com/toggle"
      }]
    }
  },
  "events": {
    "overheating": {
      "type": "string",
      "forms": [{
        "href": "https://mylamp.example.com/oh",
        "subProtocol": "LongPoll"
      }]
    }
  }
}
```

Listing 2.1: Esempio di TD serializzata con JSON-LD 1.1

Idealmente, una TD dovrebbe essere esposta dalla Thing stessa e recuperata tramite una ricerca specificia, ma non è l'unica possibilità. Alcuni dispo-

sitivi, a causa delle loro risorse limitate o non accessibili (legacy), potrebbero avere la propria TD gestita esternamente. Un pattern comune consiste nel salvare le TD in una directory che agisca da cache, velocizzandone la ricerca e facilitando la gestione dei dispositivi che rappresentano. Inoltre i vari client dovrebbero salvare la TD in una cache la prima volta che si connettono ad una Thing, modificandola soltanto in caso di un suo aggiornamento.

Per mantenere l'interoperabilità massima, le TD dovrebbero utilizzare il vocabolario specifico fornito dallo standard, il quale è già stato pensato per essere estendibile in modo chiaro e ben definito.

Riassumendo, la TD promuove l'interoperabilità in due modi:

- Consentono la comunicazione machine-to-machine nel WoT.
- Possono servire come formato comune e uniforme agli sviluppatori per documentare e recuperare tutti i dettagli necessari per accedere ai dispositivi IoT e utilizzare i loro dati.

2.3.2 WoT Binding Templates

L'IoT utilizza un'ampia varietà di protocolli per l'accesso ai dispositivi, perché ovviamente non esiste un unico approccio per ogni situazione. Proprio per questo motivo una delle sfide maggiori del WoT è sicuramente quella di riuscire ad integrare le varie piattaforme IoT (OCF,[4] oneM2M,[5] Mozilla IoT,[6] ecc.) e i dispositivi RESTful che non seguono uno standard preciso, ma che comunque forniscono un'interfaccia d'accesso HTTP o COAP. L'idea è quella di gestire questa varietà attraverso i Protocol Bindings, che devono soddisfare una serie di vincoli.

Il documento WoT Binding Templates[W3C19b] contiene una serie di "blueprint" di comunicazione che spiegano come interagire con le differenti piattaforme IoT. Ogni volta che andiamo a descrivere un certo dispositivo,

[4]https://openconnectivity.org/
[5]http://www.onem2m.org/
[6]https://iot.mozilla.org/

possiamo utilizzare i Binding Templates della corrispondente piattaforma per integrarlo, andando ad aggiungere i metadati necessari alla sua TD.

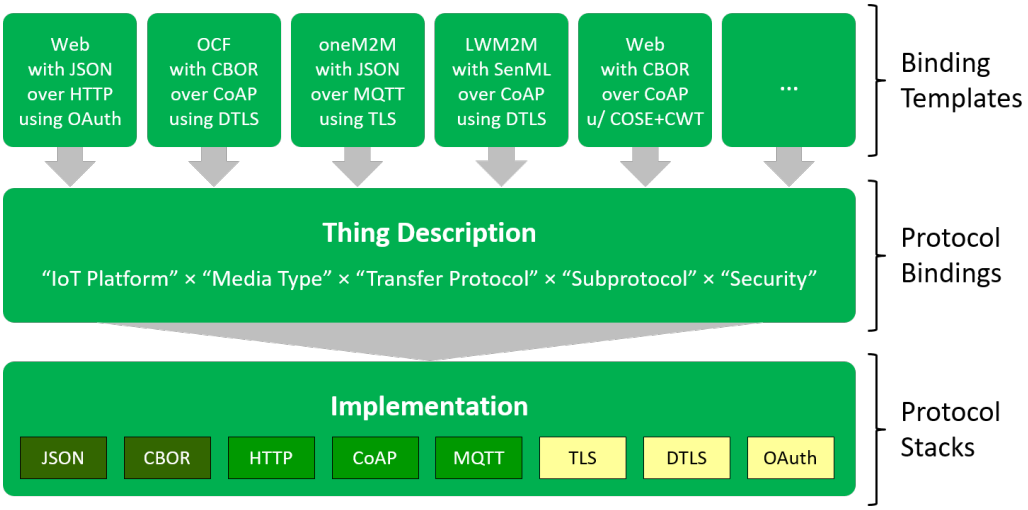

Figura 2.4: Dai Binding Templates ai Protocol Bindings [7]
fonte: [W3C19a]

I client che consumano una TD devono implementare il corrispondente Protocol Binding andando ad includere lo stack tecnologico necessario e la sua configurazione in base alle informazioni trovate nel descrittore. I metadati dei Protocol Bindings si estendono su cinque dimensioni:

- **IoT Platform**
  Le piattaforme IoT spesso introducono delle modifiche proprietarie a protocolli, quali header HTTP specifici o opzioni CoAP.

- **Media Type**
  Le piattaforme IoT differiscono spesso nei formati di rappresentazione (o nella loro serializzazione) utilizzati per lo scambio dei dati. I Media Type servono proprio per identificare questi formati.

- **Transfer Protocol**
  Si tratta dei protocolli standard dello strato Applicazione (HTTP, MQTT,

CoAP, ecc.) senza nessun tipo di opzioni specifiche o meccanismi di sotto-protocollo.

- **Subprotocol**
  I Transfer Protocol possono avere dei comportamenti particolari che richiedono di essere specificati per poterci interagire con successo. Un esempio di sotto-protocollo è la tecnica "long polling" per HTTP che ovviamente non può essere identificata tramite lo schema dell'URI.

- **Security**
  I meccanismi di sicurezza possono essere applicati a diversi livelli dello stack di comunicazione e potrebbero essere usati insieme, spesso per completarsi a vicenda.

2.3.3 WoT Scripting API

Nel documento WoT Scripting API[W3C19c] viene proposta un'interfaccia di programmazione che permette agli scritp di trovare e "consumare" una TD, istanziando degli oggetti locali che fungono da proxy per una Thing remota (con tutte le relative proprietà, azioni, eventi, ecc.). Inoltre consente agli script di implementare i vari pattern di interazione e di esporre delle Thing.

La Scripting API è un blocco opzionale del WoT, solitamente viene implementata soltanto sui nodi di rete più potenti come ad esempio i gateway. Si tratta infatti di un processo costosto da un punto di viste delle risorse necessarie per la sua esecuzione.

Queste interfacce descrivono un'API ECMAScript di basso livello che segue le specifiche della TD con precisione. Possono essere utilizzati anche altri linguaggi e runtime (ad esempio Python), l'importante è che rispettino i vincoli.

Esistono 3 elementi principali:

- **L'oggetto Wot**
  Si tratta del punto di ingresso dell'API ed è esposto da un'implemen-

tazione di WoT Runtime. Esso non espone proprietà, ma solo metodi per scoprire, consumare ed esporre una Thing.

- **L'interfaccia ConsumedThing**
  Una client API per consumare le Thing sulla rete o localmente.

- **L'interfaccia ExposedThing**
  Questa interfaccia è l'API del server che consente di definire gli handler per le richieste, le proprietà, le azioni e gli eventi su una Thing.

2.4 Implementazione del Servient

Un Servient è uno stack software che implementa gli elementi principali del WoT visti nella sezione 2.3. I Servient possono eseguire, esporre e consumare più Thing contemporaneamente; proprio per questo motivo possono agire sia come server, che come client.

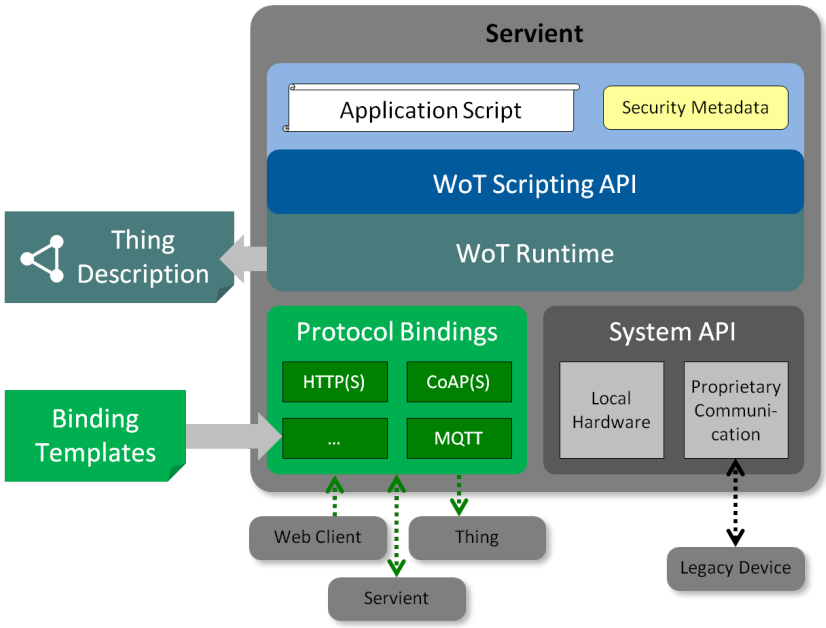

Figura 2.5: Implementazione di un Servient [8]
fonte: [W3C19a]

Nella figura 2.5 possiamo vedere l'architettura del Servient nel dettaglio. Alcuni di questi moduli li abbiamo già analizzati nella sezione precedente, ma mancano ancora alcuni tasselli fondamentali per avere il quadro completo:

- **Application**
  Le applicazioni in esecuzione sul Servient sono solitamente implementate attraverso dei file di script (ad esempio in JavaScript). Insieme agli script, devono essere forniti anche una serie di metadati che definiscono le configurazioni di sicurezza: l'ambiente di esecuzione (come devono essere isolati gli script), certificati di autenticazione e altro ancora[W3C18]. È importante sottolineare come possano esistere versioni di Servient minimali, le cui applicazioni sono sviluppate nativamente per un certo stack. In questo caso non sono presenti i moduli di Scripting API e WoT Runtime.

- **WoT Runtime**
  L'astrazione della Thing è implementata grazie al WoT Runtime che fornisce un'istanza con tutti i relativi metodi di interazione descritti nel documento WoT Scripting API. Sempre il WoT Runtime si interfaccia con i Protocol Bindings per accedere alle Thing remote e con la System API per l'hardware locale. Il sistema si occupa anche di generare la TD basata sui metadati del Servient, quelli dall'applicazione ed i Protocol Bindings disponibili.

- **Protocol Bindings**
  I Protocol Bindings sono le implementazioni dei Binding Templates. Producono i messaggi per interagire con le Thing attraverso la rete sulla base delle informazioni presenti nella TD della ConsumedThing. In molti casi, quando vengono utilizzati dei protocolli standard, è possibile utilizzare degli stack di protocolli generici. In questo caso i metadati della TD vengono utilizzati per selezionare e configurare quello corretto. Inoltre i parser e i serializzatori per i vari formati (identificati tramite i Media Type) possono essere condivisi tra i vari stack.

- **System API**
  L'implementazione di una Thing può accedere all'hardware locale o ai servizi di sistema tramite API proprietarie o altri mezzi. Un dispositivo può trovarsi fisicamente esterno al Servient, ma connesso tramite dei protocolli proprietari. In questo caso il runtime implementato può accedere al disposito legacy con questi protocolli e poi esporlo come Thing attraverso le Scripting API. Uno script può agire come gateway per il dispositivo, ma è un'opzione da considerare solo in caso non possa essere descritto tramite una TD. Questo blocco non fa parte dello standard (almeno per adesso), perché richiederebbe una grossa quantità di documentazione che sottrarrebbe tempo alla definizione degli elementi primari.

Parte II

WoT Store

Capitolo 3

Progettazione

Questo capitolo offre una panoramica della fase di progettazione della piattaforma WoT Store. In particolare, dopo aver definito gli obiettivi e i requisiti principali, viene descritta la struttura del sistema. Di ogni componente, quindi, vengono illustrate nel dettaglio le caratteristiche e le funzionalità, oltre alla descrizione del ruolo assunto all'interno dell'architettura.

3.1 Obiettivi

Lo scopo di questo lavoro è la progettazione e la realizzazione di una piattaforma per la gestione completa di applicazioni (Application) e Thing per il W3C WoT, aderendo con precisione all'architettura vista nel capitolo 2.

L'idea iniziale nasce da uno studio dell'Università di Bologna in cui viene presentato un proof-of-concept riguardante il possibile sviluppo di un'architettura di WoT Store[LC19].

Il sistema è pensato per eliminare le criticità tipiche del contesto IoT, in cui vi è la necessità di gestire dispositivi eterogenei e relativi software. È fondamentale infatti offrire un'interfaccia di ridotta complessità, che mascheri un sistema fortemente automatizzato, capace di risolvere una gran parte di operazioni che diversamente richiederebbero una grande quantità di tempo e

un'alta curva di apprendimento. Il sistema, pur richiedendo una competenza tecnica medio-elevata, può essere facilmente fruibile anche da sviluppatori non necessariamente specializzati in questo ambito. Questa caratteristica faciliterà la diffusione e l'adozione su larga scala dello standard W3C WoT, stimolando indirettamente la creazione di Thing interoperabili tra loro e la nascita di community. La sua impostazione, vista l'architettura scalabile e i meccanismi di sicurezza, lo rende utilizzabile sia in un ambiente Smart Home, sia in ambito professionale/industriale.

Da quanto esposto fin'ora, è facile notare un parallelismo tra il WoT Store e i market di applicazioni utilizzati in ambiente mobile, almeno per quando riguarda le funzionalità di base (ricerca del software da un "catalogo", download, installazione del codice direttamente dallo store). Allo stesso tempo il WoT Store introduce alcune funzionalità uniche, come l'approccio semantico e la possibilità di installare le Application contemporanemante su diversi dispositivi.

3.2 Requisiti

3.2.1 Requisiti funzionali

Il sistema deve fornire (senza escluderne altre più specifiche) le seguenti funzionalità principali:

- **Registrazione**
  L'utente deve potersi registrare inserendo i seguenti dati: email, username e password.

- **Autenticazione**
  L'utente deve poter accedere al sistema tramite le sue credenziali: email/username e password, oppure tramite un meccanismo di single sign-on come Google o GitHub.

- **Reset Password**
  L'utente potrà reimpostare la sua password inserendo l'indirizzo email con cui si è registrato.

- **Creazione Application**
  L'utente deve poter creare una nuova Application specificandone parametri semantici come nome, descrizione, tipo, visibilità, geo-localizzazione, ecc. Durante il processo di creazione, dovrà essere obbligatorio il caricamento del codice dell'applicazione. Prendiamo in considerazione due tipologie di Application differenti:

  – Thing Application (TA); forniscono un'implementazione del comportamento di una Thing coerente con la sua TD.

  – Mashup Application (MA); si rapportano con un insieme di Thing, permettendo l'interazione e l'integrazione dei dati.

- **Visualizzazione Application**
  L'utente deve poter visualizzare il dettaglio di una Application specifica: devono essere riportati tutti i metadati e un modo per recuperare il codice.

- **Creazione Thing**
  L'utente deve poter creare una nuova Thing definendone lo script, le dipendenze e in caso la configurazione del Servient su cui verrà eseguita. Questa procedura deve avvenire nel modo più semplice e automatizzato possibile, in modo tale da evitare all'utente di dover gestire manualmente un flusso di operazioni altamente complesso.

- **Visualizzazione Thing**
  La Thing deve essere processata e renderizzata leggendo la sua TD. L'utente deve poter visualizzare i valori delle sue proprietà, utilizzare le azioni disponibili e ricevere gli eventi in tempo reale. Inoltre è fondamentale la gestione dei possibili meccanismi di sicurezza e autenticazione definiti nello standard.

- **Modifica e cancellazione di Application e Thing**
  L'utente deve poter modificare o cancellare una propria Application o una propria Thing.

- **Compatibilità Application**
  Il sistema deve fornire una percentuale di compatibilità di una Application in esame con le Thing reperite, basandosi sui metadati di entrambe.

- **Installazione Application**
  Il sistema deve permettere all'utente l'installazione automatica sui dispositivi sia delle TA che delle MA, per quest'ultima dev'essere previsto però anche un ulteriore processo di deploy, così come la creazione di uno spazio hosting nel caso dovesse essere eseguita su cloud.

- **Ricerca e Scoperta**
  L'utente deve poter effettuare una ricerca semantica sulle Application e Thing contenute nel WoT Store. La ricerca può comprendere anche l'utilizzo di filtri avanzati che permettono l'aggiunta di vincoli semantici: ad esempio ricercare una Thing con un determinato additionalType "X" e geo-localizzata all'interno di una certa zona (lat, lon, radius). Oltre alla ricerca manuale, il sistema deve anche proporre una lista di Application di "tendenza", una con le nuove Application e una con le nuove Thing.

- **Condivisione**
  Le Application e le Thing devono poter essere condivise anche ad utenti specifici (ad esempio per condividere una Thing privata). L'utente deve poter visualizzare il profilo e i dati di altri utenti. L'utente deve poter decidere se seguire o abbandonare un altro utente.

3.2.2 Requisiti tecnici

Da un punto di vista tecnico, le funzionalità devono avere i seguenti requisiti aggiuntivi:

- **Supporto**
  Il portale deve poter essere accessibile tramite web e quindi compatibile almeno con le ultime versioni dei principali browser. Il deploy delle Thing, invece, deve poter avvenire su diversi tipi di dispositivi, la cui potenza di calcolo e risorse sarà spesso difficilmente paragonabile a quella di un classico computer desktop. Bisogna inoltre considerare l'eventualità che il dispositivo in questione non sia accessibile fisicamente, ma soltanto tramite rete: non è perciò possibile contare sull'utilizzo di schermi, tastiere e altre periferiche I/O che ne faciliterebbero l'installazione.

- **Sicurezza**
  La registrazione e l'autenticazione devono potersi effettuare sfruttando il protocollo OAuth2.0, oppure direttamente sulla piattaforma senza passare da altri network. Ogni risorsa deve avere un gestione completa e accurata dei permessi di accesso. Il codice delle Application dev'essere salvato, crittografato e recuperato in modo sicuro.

- **Architettura**
  L'architettura deve seguire le linee guida SOLID, rendendosi così facilmente espandibile e al contempo sicura attraverso il principio di segregazione, modulare e flessibile sfruttando l'implementazione basata su interfacce ed API e non su implementazioni.

- **Semantica**
  I metadati delle Application e delle Thing si devono basare su delle ontologie note e adeguate alla tipologia di entità che rappresentano.

3.3 Architettura

In questa sezione viene presa in esame l'architettura del WoT Store, analizzandone le diverse componenti e le diverse relazioni tra le stesse, rappresentate graficamente nella figura 3.1. Nel prossimo capitolo saranno poi forniti i dettagli implementativi.

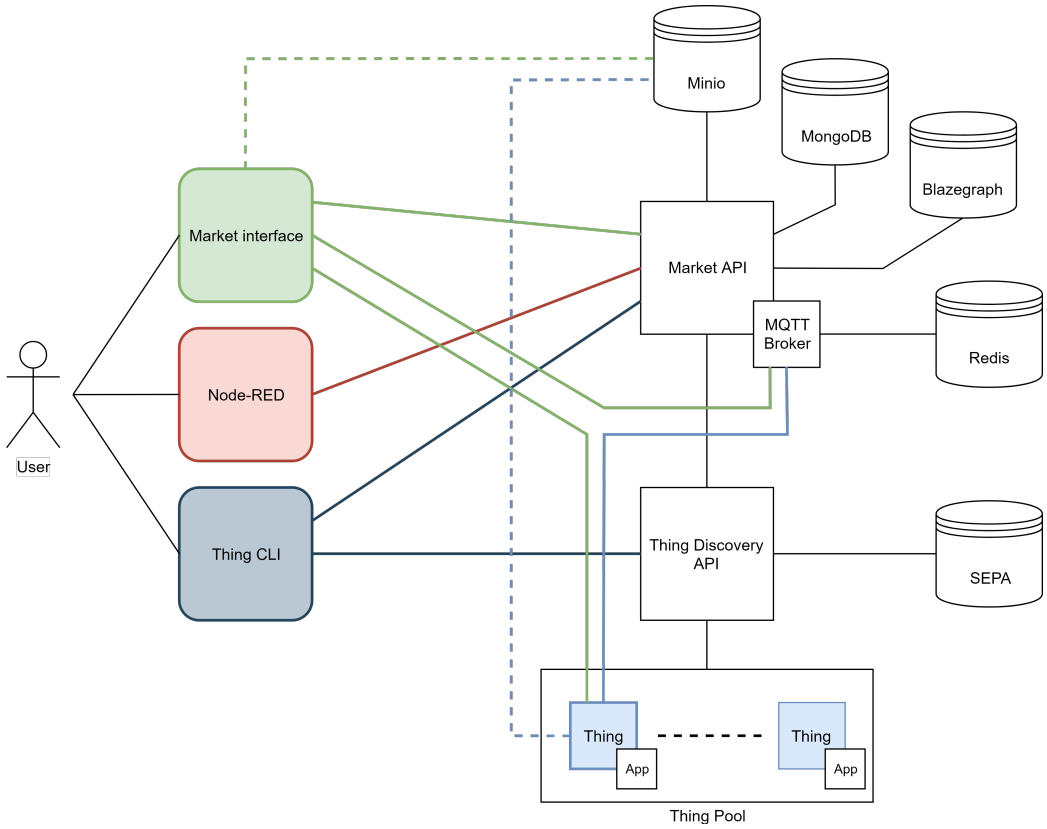

Figura 3.1: Architettura del WoT Store [1]

3.3.1 Componenti

Un primo gruppo di componenti è costituito da quegli elementi utilizzati dall'utente per interagire con i vari servizi della piattaforma:

- **Market Interface**
  La Market Interface è l'applicazione web che presenta l'interfaccia primaria dello store. Ad eccezione della creazione di una Thing, sono presenti tutte le funzionalità riportate nella sezione 3.2.1. Questa componente comunica in maniera diretta con altri 3 elementi dell'architettura: Market API (per tutte le operazioni principali sulle risorse e per ricevere eventi e notifiche); Thing (per renderizzarla in tempo reale, ottenendone i dati, quando l'utente ne apre il dettaglio); MQTT Broker (per recuperare i messaggi inviati dalla Thing).
  In caso di richiesta di download del codice di una Application, la Market Interface recupera un URL firmato dalla Market API e poi scarica il/i file dal server Minio.

- **Node-RED**
  Si tratta di uno strumento di programmazione visuale che permette di progettare flussi di informazioni, tramite il collegamento di nodi (vere e proprie unità software) tra loro. Questi nodi posso includere un'ampia varietà di dispositivi hardware, servizi online, moduli sviluppati da altri utenti. Nell'articolo [LC19] gli autori illustrano come sia stata sviluppata una serie di nodi per implementare l'architettura proposta. Questa componente non è stata ripresa durante il lavoro di tesi, ma può essere facilmente reintegrata e riallineata al nuovo sistema.

- **Thing CLI**
  La Thing CLI è uno strumento a riga di comando che aiuta l'utente nella configurazione e registrazione di una o più Thing.
  Tramite la CLI viene inizializzato un nuovo Servient sul dispositivo, generandone il file di configurazione in modo completamente interattivo. Successivamente l'utente è guidato nel deploy dei suoi script di applicazioni (oppure può usarne uno di default) e nell'assegnazione dei metadati semantici. La Thing CLI procede all'autenticazione dell'utente, e solo successivamente procede alla registrazione delle nuove Thing

sia sul servizio di discovery, sia sul server del market.

Nel secondo gruppo si trovano i tre elementi che costituiscono il back-end dell'intero sistema:

- **Market API**
  La Market API offre un'API REST per interagire con tre risorse principali: le Application, le Thing, gli utenti. Per ognuna di queste sono implementate tutte le operazioni CRUD e diverse altre chiamate per necessità più specifiche (con relativi permessi).
  Espone anche una websocket su cui si possono ascoltare alcuni eventi lanciati dal server e ricevere dati in tempo reale. Inoltre, come già spiegato precedentemente, possiede un endpoint/server MQTT che fa da broker tra gli eventi generati dalle Thing e il front-end.

- **Thing Discovery API**
  Si tratta di un servizio che opera il monitoraggio dello stato delle Thing e, grazie ad un sistema di notifiche push (pub-sub), avvisa il server centrale.
  Ogni qualvolta avviene un cambiamento (volontario o no) registra il nuovo stato della Thing, quindi lo notifica al server centrale il quale ne riaggiorna la TD e altre informazioni. Il server centrale contatta il servizio di discovery, prima di servire la TD di una Thing, per verificarne ulteriormente lo stato.

- **Thing Pool***
  Non si tratta di una vera e propria componente, ma semplicemente dell'insieme di Thing reperibili dal servizio di Thing Discovery, che può essere anche considerato come una live-directory.

3.3.2 Flussi

Per comprendere al meglio il ruolo di ogni componente, descriveremo nel dettaglio alcuni dei flussi delle funzionalità principali del sistema:

- **Ricerca semantica**

  1. L'utente, attraverso l'interfaccia web dello store, inserisce il nome della Application che sta cercando ed eventuali filtri aggiuntivi.
  2. I parametri vengono inviati al server, il quale, attraverso una specifico connettore realizzato ad hoc, costruisce la query SPARQL e recupera tutte le Application che soddisfano i criteri richiesti.
  3. I risultati sono restituiti in formato JSON-LD e visualizzati sotto forma di lista sull'interfaccia.

- **Creazione di Application**

  1. L'utente compila il form comprensivo di tutte le proprietà semantiche e fornisce il/i file col codice sorgente dell'applicazione (un singolo file .js o uno .zip).
  2. La richiesta viene processata dal server, il quale salva il codice su Minio, i metadati su Blazegraph e la relazione con l'utente su MongoDB.
  3. Viene restituita la nuova risorsa creata in formato JSON-LD.

- **Creazione di Thing (tramite CLI)**

  1. All'inizio viene richiesto all'utente se vuole utilizzare i propri script oppure quello di default.
  2. Nel primo caso gli verrà fornito un nuovo UUID, che dovrà copiare come @id della sua nuova Thing; quindi gli verrà chiesto di inserire il nome della Thing (indispensabile per ottenere successivamente la sua TD).

3. Nel secondo caso sarà richiesto l'inserimento di tutti i metadati semantici per la generazione di una Thing vergine (con Application di default).
4. Lo strumento permette la creazione simultanea di più Thing, per cui gli step 2 o 3 verranno ripetuti finché l'utente non deciderà di interrompere il processo.
5. A questo punto verrà richiesto all'utente di scegliere se configurare le opzioni del Servient manualmente, oppure utilizzare le impostazioni di default.
6. Viene poi inizializzato ed avviato il Servient (i cui dettagli verranno riferiti nel prossimo capitolo), il quale espone le nuove Thing.
7. Vengono quindi richieste all'utente le proprie credenziali relative al WoT Store (con le quali viene recuperato il token di autenticazione).
8. Infine, tramite una HTTP GET locale, sono recuperate le TD delle nuove Thing, che vengono poi pubblicate sul servizio di discovery e sul WoT Store.

- **Aggiornamento di un gruppo di Thing**

1. Inizialmente l'utente imposta una serie di filtri per circoscrivere la ricerca delle Thing su cui vuole installare l'Application scelta.
2. La Market Interface effettua una chiamata HTTP GET /things, passando i parametri come filtri, alla Market API.
3. Il server del WoT Store recupera dal proprio Blazegraph le Thing, dopodiché interroga il servizio di discovery per sapere quali sono attive (live).
4. I risultati vengono ritornati al front-end così da permettere all'utente di selezione le Thing su cui vuole effettuare il deploy.

5. Gli ID delle Thing selezionante vengono inviati al server, il quale genera degli URL firmati e li invia alle Thing corrispondenti.
6. Ogni Thing scarica il codice della Application da Minio e lo esegue.
7. Il servizio di discovery notifica il server centrale ogni volta che viene completato un aggiornamento, il quale invia gli eventi all'applicazione web per fornire un feedback e i nuovi dati all'utente.

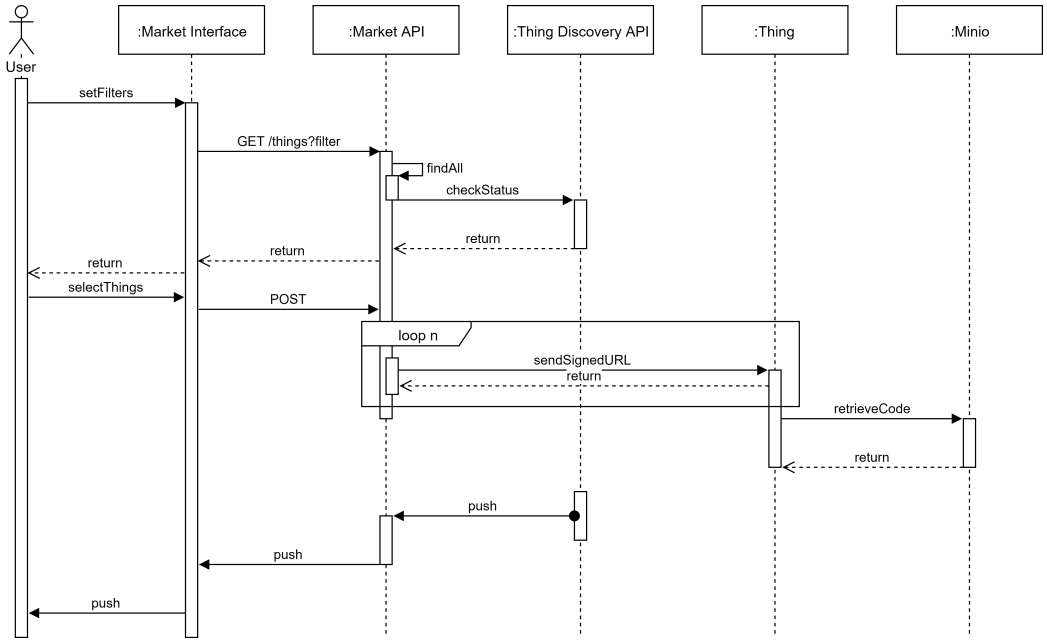

Figura 3.2: Sequenza di azioni invocate durante le operazioni di aggiornamento di un gruppo di Thing [2]

Capitolo 4

Implementazione

Lo sviluppo del sistema ha richiesto l'integrazione di diverse componenti al fine di fornire le funzionalità illustrate precedentemente. In questo capitolo vengono descritte le tecnologie utilizzate e i moduli realizzati, focalizzando l'attenzione su quelli che sono stati i problemi implementativi più significativi e su come sono stati risolti.

4.1 Tecnologie

Trattandosi di un progetto "full-stack" sono risultate necessarie numerose tecnologie: in questa sezione verranno riportate soltanto quelle principali, con una breve discussione sulle motivazioni che mi hanno portato ad effettuare certe scelte piuttosto che altre.

Il server del WoT Store è composto da un'applicazione per Node.js v10.x che utilizza il framework LoopBack v3 (LTS) e un Broker MQTT realizzato con la libreria Mosca. Per funzionare ha bisogno dei seguenti database/storage:

- Minio: un object storage server per salvare i sorgenti delle Application.
- MongoDB: un database che viene utilizzato per la gestione ordinaria degli utenti e delle relazioni tra i vari modelli.

- Blazegraph: un triplestore e graph database in cui il server salva le TD delle Application e delle Thing.

- Redis: una database key-value (in-memory), altamente performante, per il Broker MQTT.

Il client web è sviluppato in Angular v6.x ed utilizza una serie di pacchetti dei quali citerò solo i più importanti: @angular/material, per avere componenti in stile Material Design già pronti; ngx-translate, per le traduzioni multilingua; @hyperloris/tyson, per convertire i JSON in istanze di classi TypeScript e viceversa; ngx-mqtt, per avere un client MQTT; socket.io-client, per la comunicazione in tempo reale client – server.

Il Thing CLI è a sua volta un'applicazione Node.js e sfrutta pacchetti come chalk, clui, inquirer e altri per la costruzione del terminale interattivo; request e shelljs per le chiamate HTTP e i vari comandi necessari.

4.1.1 Node.js

Uno dei motivi che mi ha spinto ad utilizzare Node.js è sicuramente la sua community che negli anni ha prodotto una grandissima quantità di software open-source, arrivando a rendere disponibili centinaia di migliaia di moduli.[1]

Se vogliamo analizzare la scelta da un punto di vista ancora più tecnico, ci possiamo soffermare su due caratteristiche che lo rendono estremamente efficiente in questo tipo di contesto:

- non-blocking I/O: questo modello è pensato proprio per gestire le operazioni in background, infatti tutte le richieste di I/O sono processate su un thread separato rispetto a quello principale di esecuzione. Ciò significa che un file particolarmente grande (o problematico) in fase di lettura non ostacolerà le prestazioni del programma principale.

[1] http://www.modulecounts.com/

- Garbage Collection: dovendo manipolare degli array di grandi dimensioni, Node.js (l'engine V8 su cui si basa) torna particolarmente utile per via della sua pulizia automatica della memoria[Kha17].

4.1.2 LoopBack

Il framework LoopBack consiste in una serie di moduli Node.js che possono essere utilizzati insieme o separatamente per costruire delle API REST. Viene sviluppato e mantenuto da IBM / StrongLoop e proprio recentemente è stata rilasciata la quarta versione interamente scritta in TypeScript. Quando ho iniziato questo progetto il suo stato non era ancora abbastanza maturo e per questo motivo ho optato per rimanere sulla terza.

Sono presenti numerose funzionalità out-of-the-box come: autenticazione base, generazione degli endpoint CRUD, API Explorer e molto altro.
Uno degli aspetti sicuramente più interessante è la parte riguardate i modelli dati:

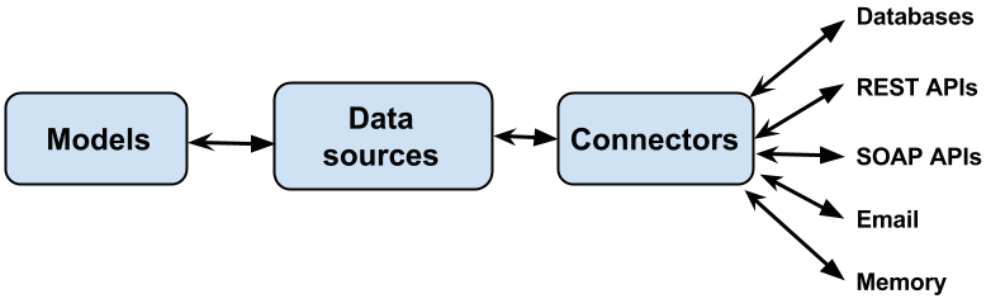

Figura 4.1: Architettura LoopBack per i modelli dati [2]

LoopBack generalizza i servizi di back-end come database, API REST, servizi di storage, ecc. L'applicazione, una volta definiti i modelli, parla solo ed esclusivamente con le API esposte da DataSource[3] e PersistedModel,[4] le

[3]https://apidocs.loopback.io/loopback-datasource-juggler/#datasource
[4]https://apidocs.loopback.io/loopback/#persistedmodel

quali, a loro volta, comunicano con i connettori. In questo modo tutto il sistema rimane ad un livello di astrazione superiore e lo sviluppatore non si deve preoccupare di come i dati verranno poi salvati e recuperati dai vari database/servizi. Per questo lavoro è stato realizzato un connettore completo per il triplestore Blazegraph come vedremo nella sezione 4.2.1.

4.1.3 Angular

Angular è un framework open-source scritto in TypeScript per lo sviluppo di applicazioni Web. È stato sviluppato principalmente da Google ed è l'evoluzione (completamente rifatto) del suo predecessore AngularJS.

Angular fornisce, per una applicazione web di complessità medio-alta, una architettura sufficientemente strutturata da garantire il massimo livello di riutilizzabilità e modularizzabilità del codice. Questo attraverso una componentizzazione spinta dei componenti grafici e delle strutture di connessione con i vari tipi di back-end REST. Inoltre, attraverso un opportuno uso di strumenti oramai integrati nell'infrastruttura Angular, quali Angular Elements[5] e gli strumenti di librarizzazione, diventa molto semplice trasformare questi componenti iper-ingegnerizzati specifici di Angular in Web Component standard, rendendoli disponibili per tutti gli altri framework in modo nativo o addirittura come elementi di base per applicativi nuovi che riutilizzino anche solo in parte i dati REST.

4.1.4 Docker

Docker è uno strumento che può pacchettizzare un'applicazione e le sue dipendenze in un container virtuale che può essere eseguito su qualsiasi server Linux. L'intera piattaforma (client, server e tutti i database/storage) sono stati preparati per un deploy su un cluster Docker Swarm[6] come Stack, ovvero una serie di Service interconnessi tra loro. Il vantaggio di questa scel-

[5]https://angular.io/guide/elements

[6]https://docs.docker.com/engine/swarm/

ta è triplice: rende il sistema facilmente scalabile e fault-tolerant grazie alla capacità di replicare i servizi su diverse macchine del cluster; offre lo spazio per le MA; consente una gestione rapida e puntuale dell'infrastruttura.

4.1.5 Eclipse Thingweb node-wot

Per la realizzazione del sistema è necessaria un'implementazione di Servient da installare sui dispositivi. Attualmente l'unica supportata e garantita dal W3C[W3C19a] è quella sviluppata dalla Eclipse Foundation: Eclipse Thingweb node-wot.[7]

Si tratta di un Servient sviluppato per Node.js che richiede alcune dipendenze aggiuntive per essere eseguito: Python 2.7, make e un compilatore C/C++ (tipo GCC).

4.2 Moduli

4.2.1 Server

I file del server sono suddivisibili in 4 categorie principali:

- **Configurazione**
  Attraverso questi file è possibile gestire la configurazione di alcuni componenti fondamentali come l'explorer delle API, le datasource (url, porte, connettori, ecc.), i middleware (compressione, opzioni di CORS, helmet,[8] ecc.), i modelli (datasource utilizzata e visibilità, ovvero se una determinata risorsa viene esposta pubblicamente oppure no).

- **Boot**
  I file di boot contengono quelle operazioni che devono precedere l'attivazione delle API: creazione della cartella di storage e del bucket su Minio, abilitazione dell'autenticazione, creazione dei ruoli per gli utenti, inizializzazione del broker MQTT.

[7] https://github.com/eclipse/thingweb.node-wot
[8] https://helmetjs.github.io/

- **Mixin**
  I mixin sono sostanzialmente delle funzionalità ”generiche” che vengono sviluppate in modo tale da renderle applicabili a qualsiasi modello. In questo lavoro sono stati implementati tre mixin: il primo aggiunge al modello le proprietà e gli endpoint necessari per trasformarlo in una risorsa su cui è possibile applicare una ”star” (es. PUT /applications/:id/star e DELETE /applications/:id/star), il secondo aggiunge e gestisce i timestamp di creazione e aggiornamento, mentre il terzo permette di annotare una proprietà in modalità ”solo lettura”.

- **Modelli**
  I modelli sono il fulcro dell'applicazione server: ogni risorsa (es. Application, Thing, User, ecc.) ha un file JSON in cui vengono descritte le proprietà, le relazioni, i permessi. Oltre a questo è presente un file JavaScript in cui viene implementata la business logic della risorsa stessa.

LoopBack Connector Blazegraph

Una parte significativa del lavoro lato server è consistita nello sviluppo di un connettore per Blazegraph. Per sviluppare un connettore LoopBack è necessario implementare una serie di metodi che permettano al framework di interfacciarsi col database:

- create
- updateOrCreate
- replaceOrCreate
- findOrCreate
- buildNearFilter
- all
- destroyAll
- count
- save
- update

- destroy
- replaceById
- updateAttributes

A questo punto il framework è in grado di gestire autonomamente l'inizializzazione, lifecycle, migrazioni, operazioni CRUD, ecc.

Le maggiori difficoltà riscontrate durante la realizzazione di questa componente, sono state la mappatura dei filtri inviati dal client in query SPARQL e la conversione dei formati dati.

Per quanto riguarda la seconda problematica, bisogna considerare il caso in cui i dati vengono inviati dal server al Blazegraph, dove il formato deve passare da JSON-LD a N-Quads. In questa situazione viene utilizzata una libreria chiamata jsonld.js[9] che permette di espandere il JSON-LD e successivamente di trasformalo in quadruple. Quando invece i dati vengono recuperati dal triplestore, otteniamo un JSON che deve essere integrato con ulteriori metadati e riportato al suo stato di JSON-LD, per essere conforme ai modelli e quindi a tutto il sistema.

4.2.2 Client

L'architettura del client è sicuramente quella più complessa. Per riuscire a districarci al meglio, riporto un'immagine che ne mostra la struttura dei moduli.

[9]https://github.com/digitalbazaar/jsonld.js

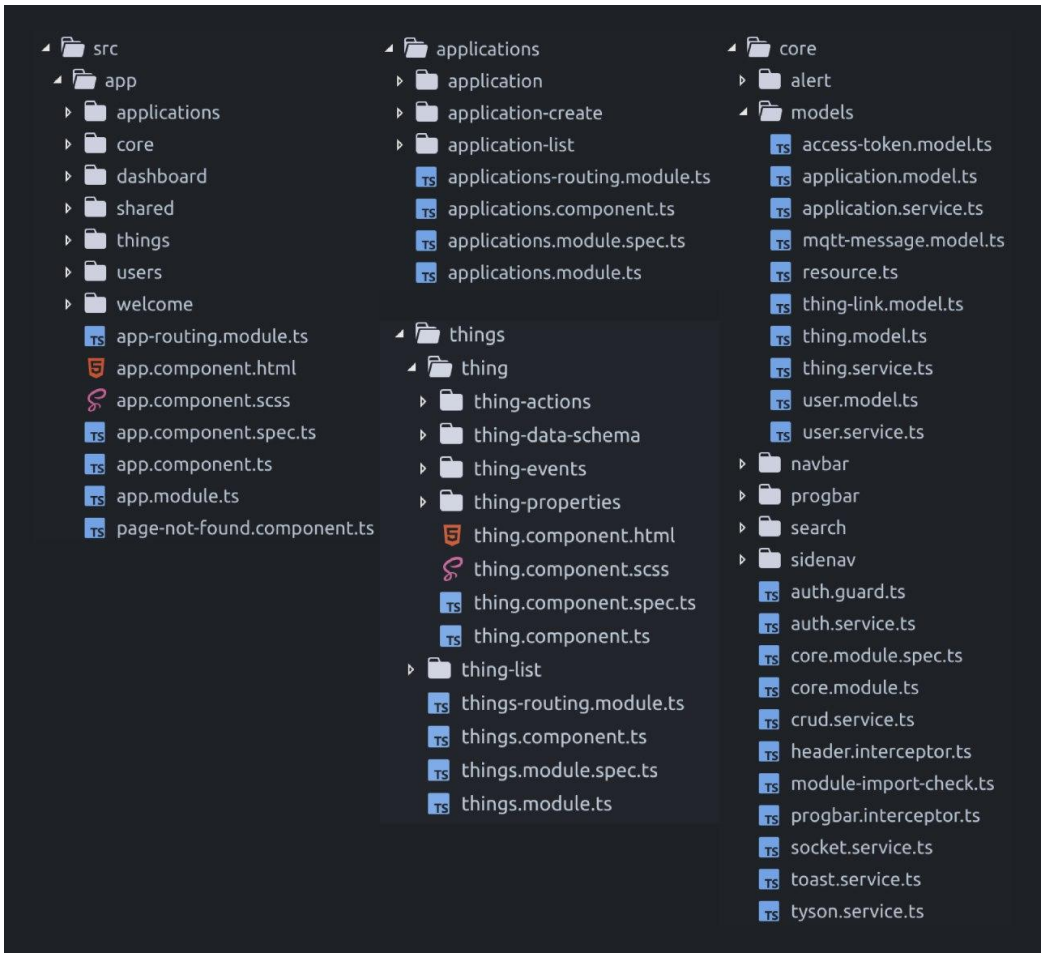

Figura 4.2: Struttura dei moduli dell'applicazione client [10]

L'applicazione è fortemente modulare ed ogni risorsa ha il suo set di componenti che ne costruisce una visualizzazione a lista, una visualizzazione a dettaglio, un form di creazione e modifica. Il modulo core contiene i modelli dati e i loro relativi servizi, i quali ereditano da un servizio REST CRUD generico, estendendone le chiamate e le funzionalità. Inoltre offre i componenti globali che costituiscono lo scheletro dall'applicazione web, come la barra di navigazione e la barra laterale. Infine sono presenti elementi riguardati l'autenticazione e la sicurezza, come auth.guard per proteggere alcuni endpoint dalle sessioni non autenticate, auth.service per la gestione dei token, header.interceptor che manipola gli header HTTP in uscita.

Una nota di riguardo per l'implementazione del protocollo di websocket che in combinazione con la libreria Tyson, in caso di aggiornamento di una risorsa, permette di inviare la sola parte modificata ed integrarla automaticamente nell'istanza esistente.

Una particolare attenzione va riservata a quei componenti che contribuiscono alla renderizzazione della Thing a partire dalla sua TD. Tramite una serie di factory il ThingComponent e i suoi sottocomponenti generano tutta l'interfaccia grafica, visualizzandone le proprietà e i relativi valori; costruiscono i form per l'interazione con le azioni; mostrano gli eventi in tempo reale. L'approccio tramite factory garantisce una facile espandibilità sia dal punto di vista grafico, che da quello dei protocolli di comunicazione. Allo stesso modo vengono gestiti anche i meccanismi di sicurezza che si possono trovare su una TD del W3C.

4.2.3 Thing CLI

Il Thing CLI è composto da 6 file principali:

- **cli**: funzioni necessarie per stampare a video i vari "elementi grafici", come il titolo, le descrizioni e i divisori.
- **files**: funzioni di utilità per la cancellazione e scrittura dei file su disco.
- **inquirer**: contiene tutte le domande (alberi decisionali) dei vari passaggi richiesti.
- **sepa**: codice per la comunicazione con il servizio di discovery.
- **servient**: inizializzazione del Servient (vedi sotto), generazione del file di script di default, creazione del file di configurazione, ecc.
- **wots**: comunicazione con l'API del WoT Store, per l'autenticazione, registrazione delle nuove Thing, ecc.

Qui di seguito verrà descritto come avviene la fase di inizializzazione del Servient su un dispositivo "vergine". Per prima cosa la repository del Eclipse

Thingweb node-wot viene clonata tramite git e riportata al commit impostato nel file di configurazione del CLI. Il reset è necessario, perchè il framework si trova in una fase di sviluppo molto intensa e spesso vengono introdotti dei breaking-change che ne causerebbero il malfunzionamento. A questo punto avviene l'installazione di pm2,[11] dei pacchetti npm e viene lanciata la build. Infine si esegue il Servient tramite pm2, passandogli come parametri il file di configurazione e il path in cui si trovano gli script delle applicazioni.

```
1  const init = async () => {
2    const status = new Spinner('Initializing the framework,
       please wait...');
3    status.start();
4
5    const commands =
6      `git clone ${config.framework.repoUrl} && ` +
7      `cd ${config.framework.folderName} && ` +
8      `git reset --hard ${config.framework.commit} && ` +
9      'npm install pm2@latest -g && ' +
10     'npm install && ' +
11     'npm run build && ' +
12     `pm2 start packages/cli/dist/cli.js -- --configfile ${
         appRoot}/scripts/wot-servient.conf.json ${appRoot}/
         scripts/ *`;
13
14   if (shell.exec(commands, {silent: true}).code !== 0) {
15     throw new Error('Installation failed.');
16   } else {
17     log(chalk.green('OK'));
18     status.stop();
19   }
20 };
```

Listing 4.1: Funzione di inizializzazione del Servient

[11] http://pm2.keymetrics.io/

Capitolo 5

Validazione

In questo capitolo viene riportato un test condotto per verificare l'applicabilità del sistema a quello che potrebbe essere un caso d'uso reale. Si vuole dimostrare come il WoT Store rappresenti una piattaforma utile al monitoraggio e alla gestione dello stato di reti eterogenee e delle applicazioni che le utilizzano. Inoltre viene valutata l'adeguatezza del Thing CLI quale strumento dedicato alla rapida e standardizzata installazione di Servient e nuove Thing.

5.1 Configurazione

Per la realizzazione delle varie reti sono state utilizzate le seguenti tecnologie hardware:

- **NodeMCU**
  Si tratta di una piattaforma open source creata specificatamente per l'IoT. Possiede un chip ibrido Wi-Fi 802.11 b/g/n e Bluetooth v4.2, 128KB di memoria e 4MB di storage.

- **Raspberry Pi 3 Model B+**
  Un computer single-board progettato per ospitare sistemi operativi basati sul kernel Linux. Ne esistono varie versioni che si distinguono

dalla loro configurazione hardware. Il Raspberry Pi 3 Model B+ è equipaggiato con un processore ARM 64-bit quad-core da 1.4 GHz, 1 GB di memoria LPDDR2 SDRAM, un adattatore Ethernet, uno Wi-Fi 802.11.b/g/n/ac e uno Bluetooth v4.2. È sicuramente la board più performante dell'intera infrastruttura.

- **Arduino Uno Rev3 con Shield - Xbee**
  Un microcontrollore dotato di 16 MHz di velocità di clock, 2KB di SRAM e 32 KB di storage. In questo caso è stato necessario aggiungere un modulo chiamato Shield - Xbee per permettere alla board di comunicare utilizzando Zigbee.

- **DHT11**
  Un sensore di temperatura e umidità con uscita dei dati in formato digitale.

Sono state predisposte 3 reti di sensori differenti, ognuna delle quali contiene dei nodi che parlano con una propria tecnologia (figura 5.1).

Per quanto riguarda il software, sono state realizzate delle System API (implementate in JavaScript) strutturate su due livelli: Device Query che ha il compito di gestire la comunicazione request-response con gli end-node, IPC che rende i dati disponibili alla Scripting API del Servient. Le System API, fungendo da tramite, rendono possibile la comunicazione dei microcontrollori col Servient, secondo queste 3 tipologie di standard di trasmissione dati: Wi-Fi (UDP), seriale, Bluetooth Low Energy (BLE).

La rete A è composta da 3 NodeMCU con sensori DHT11 e un Raspberry Pi. Tramite Wi-FI, i NodeMCU sono connessi su una LAN secondaria, mentre il Raspberry Pi è connesso sia sulla LAN secondaria, sia su quella principale. I nodi raccolgono i dati dai sensori e li mandano in broadcast sulla LAN tramite UDP. La System API dedicata ascolta i messaggi inviati sulla rete e tramite ZeroMQ (una libreria per la comunicazione inter-process) li inoltra al Servient, che legge i pacchetti e li invia alla Thing specificata nel messaggio. La rete B contiene 3 Arduino con modulo Xbee, il primo configu-

rato come ZigBee Coordinator (ZC), gli altri due come ZigBee End Device (ZED). In questo caso il Raspberry Pi è in rete mediante Ethernet. Il ZC comunica tramite seriale con la System API, che poi si comporta come nel caso precedente. La rete C è formata da 3 NodeMCU BLE e un Raspberry Pi, come il precedente cablato. Diversamente dalle altre due situazioni, i nodi comunicano direttamente con la System API. Il resto dello stack è gestito come nelle reti A e B.

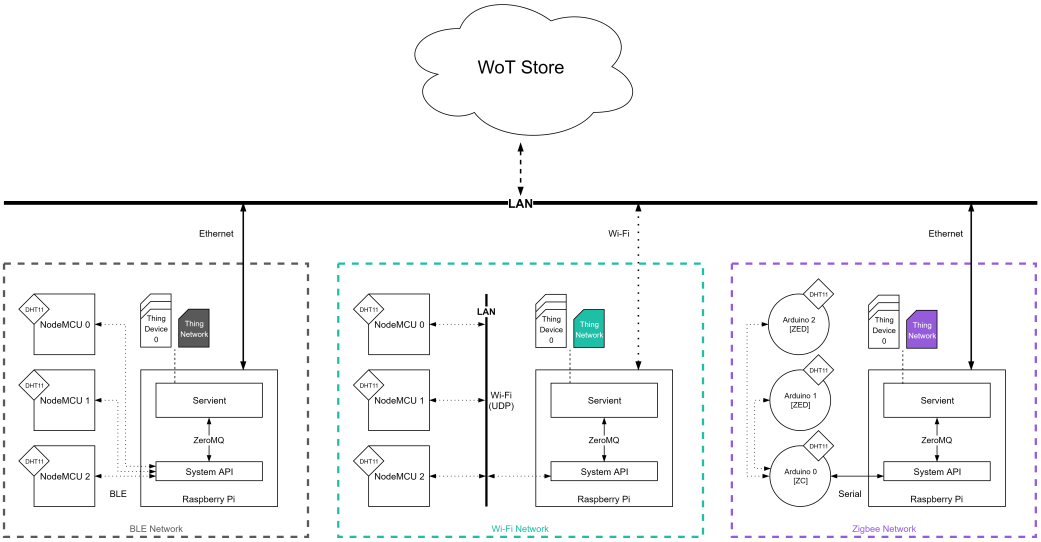

Figura 5.1: Configurazione delle reti per il test [1]

Su tutti i Raspberry Pi sono stati installati Raspbian Stretch Lite (un'immagine minimale basata su Debian Stretch), Node.js e lo strumento Thing CLI.

Per mezzo del Thing CLI sono stati generati gli script di default con i metadati specifi per ogni nodo, inizializzati e configurati i Servient, registrate le Thing. Ogni Servient espone due tipi di Thing: Thing Device che descrive le proprietà, le azioni e gli eventi degli end-node, e Thing Network che descrive la performance complessiva della propria rete di sensori. Inoltre, consideriamo tre possibili Protocol Bindings per ogni Thing: HTTP, CoAP e MQTT.

Quindi, tramite il WoT Store, abbiamo installato le TA specifiche per i gruppi di nodi e lanciato il deploy di una MA. Nella figura 5.1 viene mostrato un frammento della TD di una Thing Device. La TD include una azione (*getData* per il polling dei dati), due proprietà (*frequency* per la frequenza di campionamento e *state* per lo stato del link, che può essere connesso o disconnesso) e due eventi (*newData* e *newState*, rispettivamente lanciati quando il sensore invia nuovi dati o quando c'è un cambiamento nello stato del link di comunicazione). La TD della Thing Network include una serie di proprietà che descrivono la performance della rete: il delay, il packet delivery ratio e il throughput calcolato fino a quel momento.

```
{
  "@context": [
    "http://www.w3.org/ns/td",
    { ... }
  ],
  "@type": "Thing",
  "@id": "urn:uuid:fe7edf6f-e5d8-4626-9650-6b29dc0877e3",
  "name": "teramo-zigbee1",
  ...
  "properties": {
    "state": { ... },
    "frequency": { ... }
  },
  "actions": {
    "getData": { ... },
    "setFrequency": { ... }
  },
  "events": {
    "newData": { ... },
    "newState": { ... }
  },
  ...
}
```

Listing 5.1: TD di una Thing Device

La figura 5.2 mostra un frammento di codice della MA che, consuma tutte

le Thing della rete e, attraverso un algoritmo round-robin raccoglie i dati in maniera non selettiva, quindi produce un file CSV, utilizzato per tracciare i grafici successivi.

```
...

WoT = await servient.start();
for(const name in DOMAIN_PER_THINGS) {
  const network = NETWORK_PER_THINGS[name];
  if(!(network in N_DEVICES_PER_NETWORKS)) {
    N_DEVICES_PER_NETWORKS[network] = 0;
  }
  N_DEVICES_PER_NETWORKS[network]++;
  const domain = DOMAIN_PER_THINGS[name];
  const url = 'http://' + domain + ':' + port + '/' + name;
  const thing = await consumer.consumeThing(WoT, url);
  thing.getClients().set('http', http_client);
  thing.getClients().set('coap', coap_client);
  THINGS[name]= thing;
}

...

const res = await race(thing.actions['getData'].invoke());

...
```

Listing 5.2: Codice della MA per la raccolta dati

Si può notare come, per mezzo degli strumenti forniti dalla piattaforma WoT Store e grazie all'architettura W3C WoT, sia stato possibile gestire in maniera completamente automatizzata un sistema di reti di sensori eterogenee, astraendosi dai dettagli tecnologici e prescindendo dalla struttura fisica della rete.

5.2 Risultati

In questa sezione mostriamo graficamente i dati che sono stati raccolti e analizzati dalla MA, provenienti da più di 7000 misurazioni.

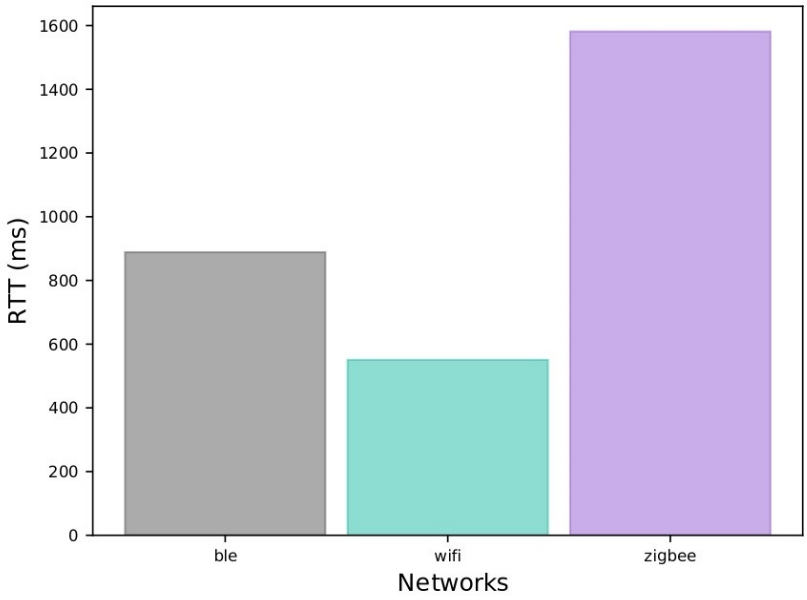

Figura 5.2: Round Trip Time medio delle tre reti [2]

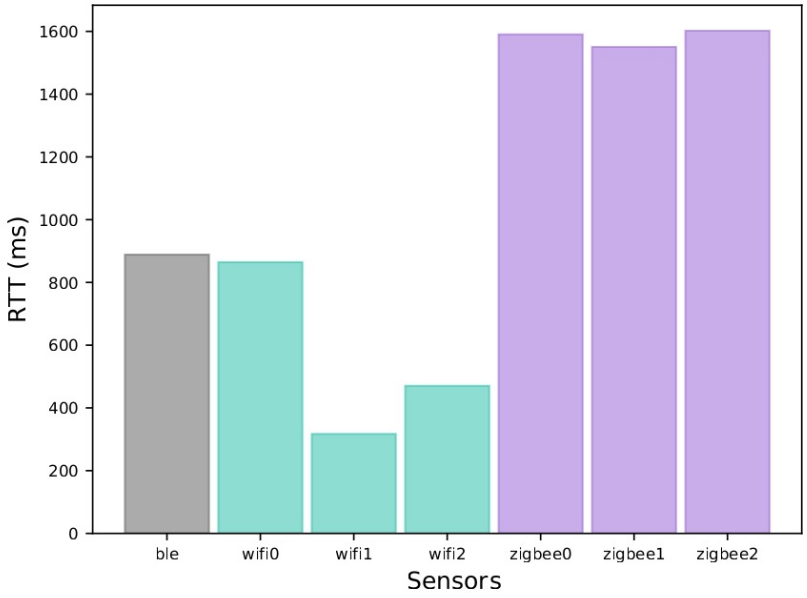

Figura 5.3: Round Trip Time per ogni singolo sensore [3]

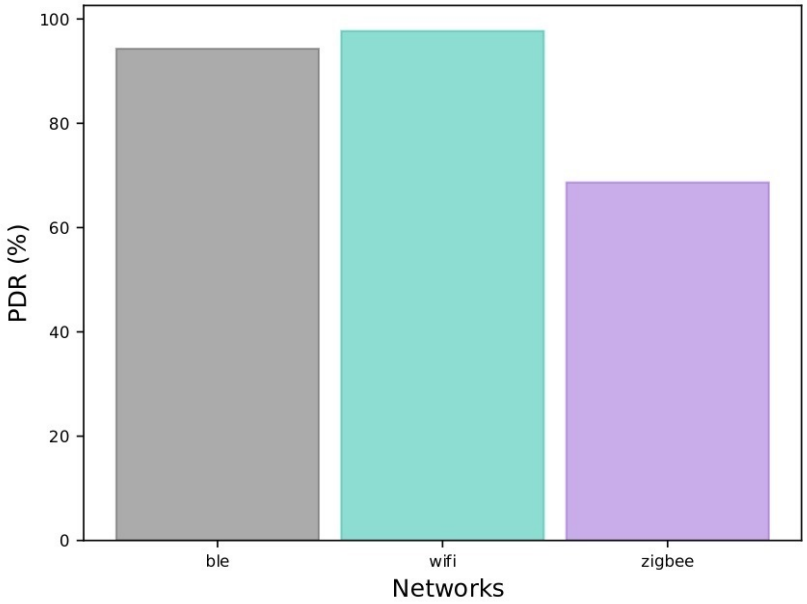

Figura 5.4: Packet Delivery Ratio medio delle tre reti [4]

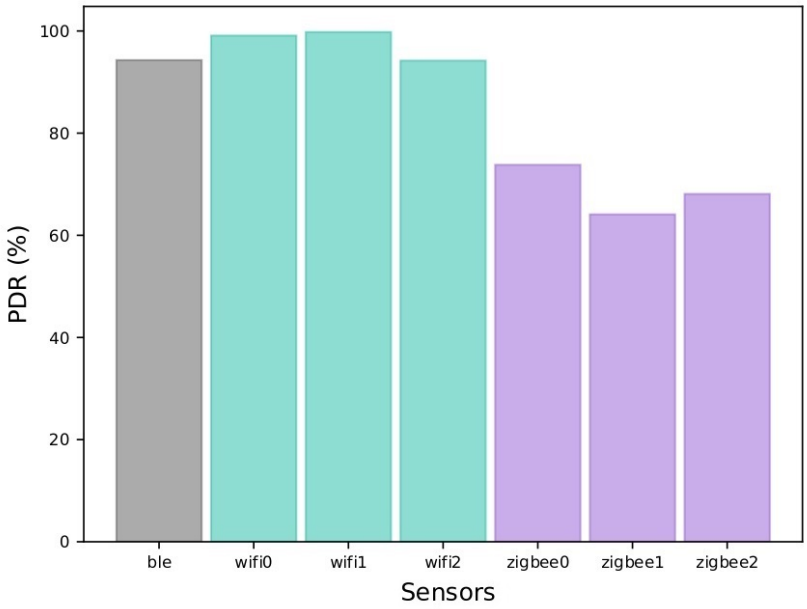

Figura 5.5: Packet Delivery Ratio medio delle tre reti [5]

5.3 Schermate

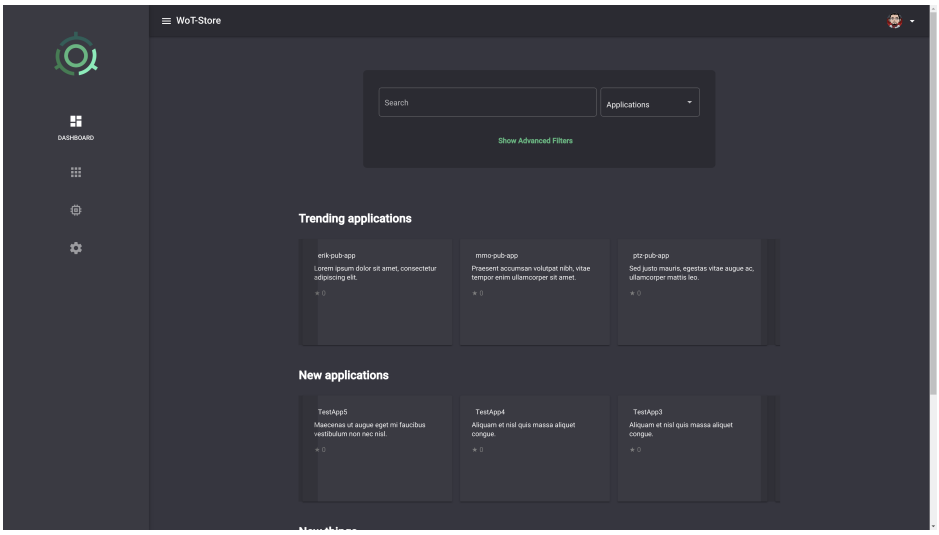

Figura 5.6: Dashboard del WoT Store [6]

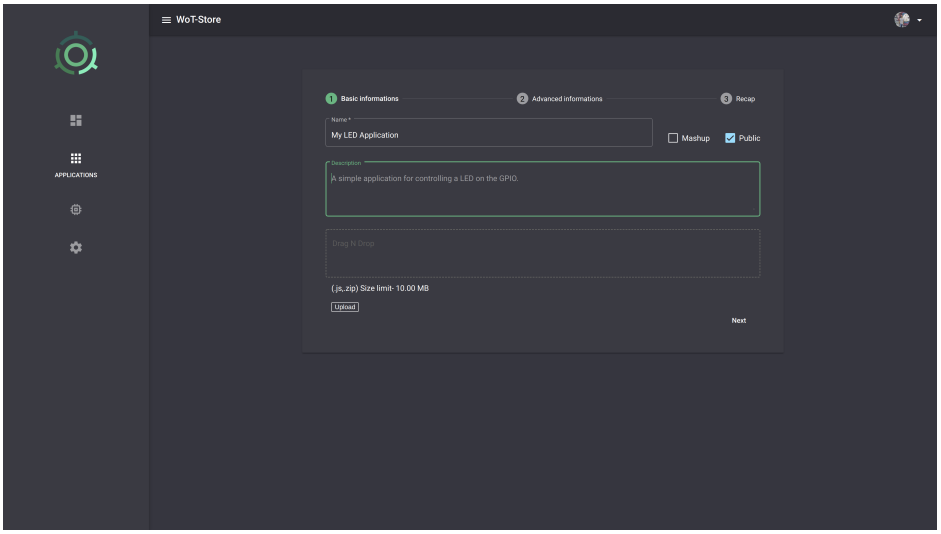

Figura 5.7: Schermata di creazione di una nuova applicazione [7]

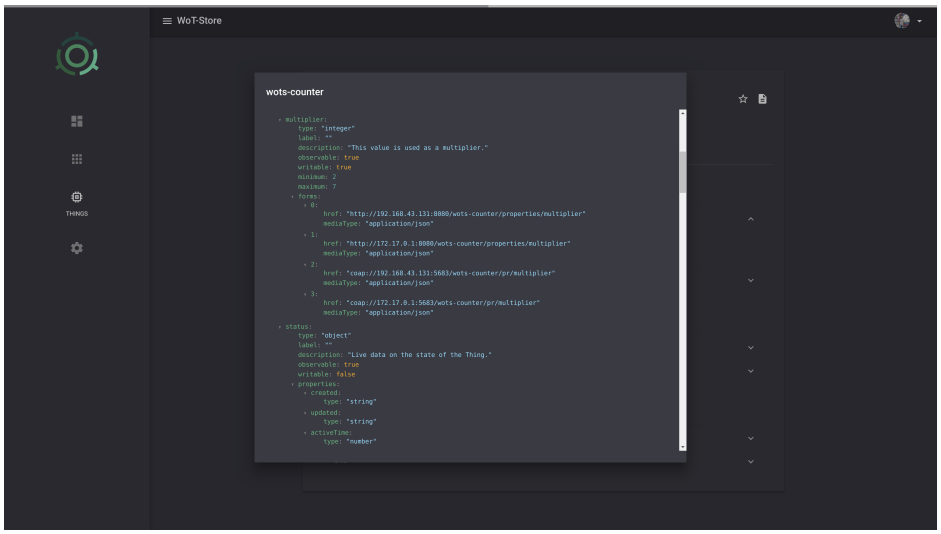

Figura 5.8: Dialog in cui viene mostrata la TD di una Thing [8]

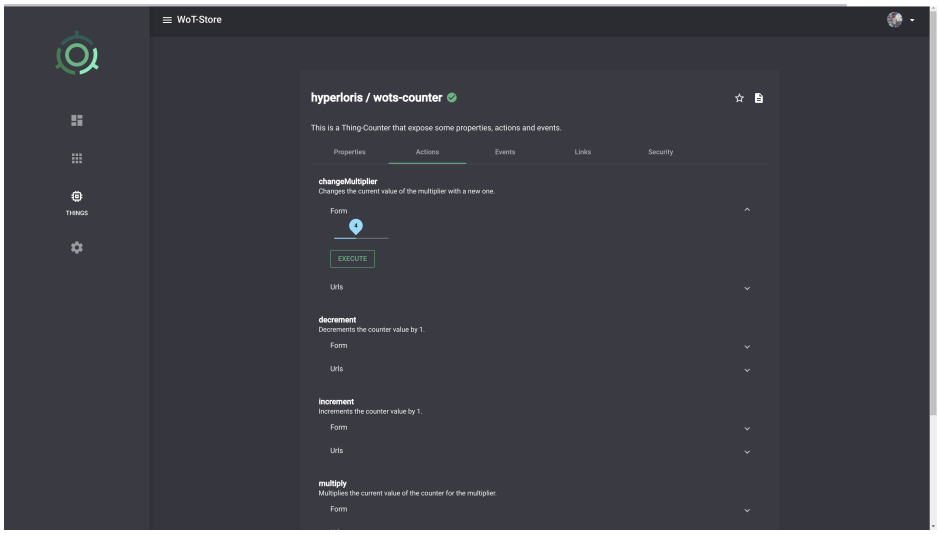

Figura 5.9: Schermata di dettaglio di una Thing (azioni disponibili). [9]

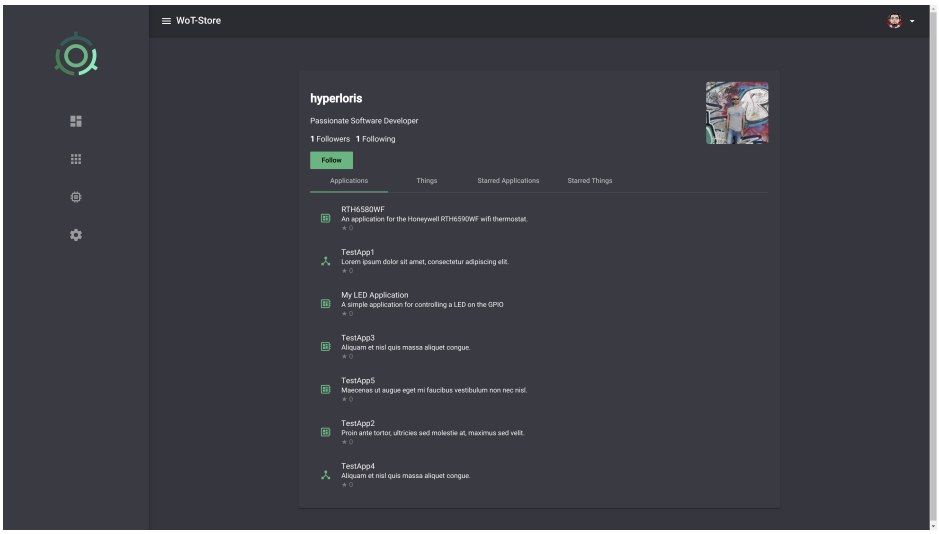

Figura 5.10: Schermata del profilo utente. [10]

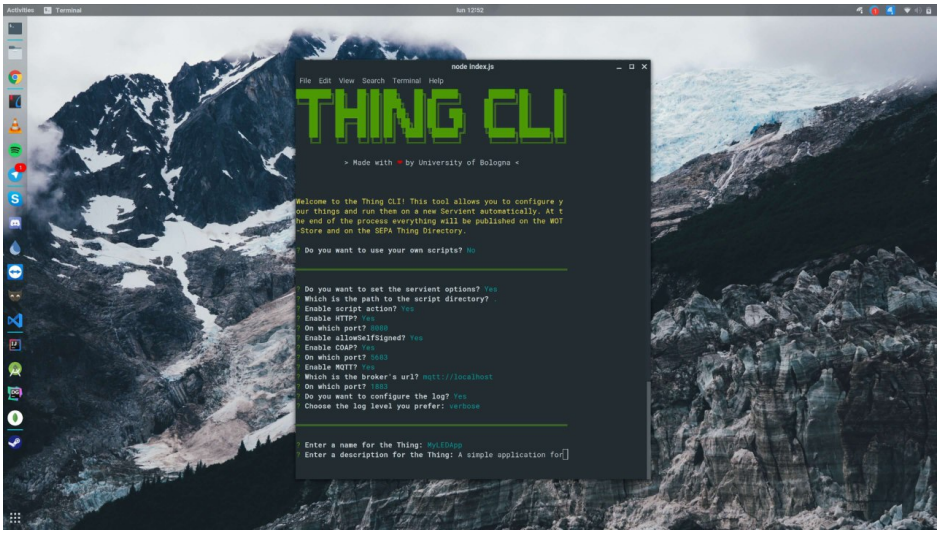

Figura 5.11: Configurazione delle impostazioni del Servient tramite Thing CLI. [11]

Figura 5.12: Configurazione dei propri script tramite Thing CLI. [12]

Capitolo 6

Conclusioni

In questo elaborato è stata presentata la realizzazione di una piattaforma per la gestione di applicazioni e thing basate sullo standard proposto dal W3C.

Per prima cosa era necessaria una fase preliminare di studio degli aspetti tecnologici relativi alla situazione attuale del mondo IoT e WoT, per poi passare ad un'analisi dettagliata dell'architettura proposta dal W3C. A ciò è seguita la fase di progettazione e implementazione delle principali componenti che costituiscono il sistema nella sua interezza.

Il lavoro svolto si è rivelato molto lungo e impegnativo, infatti si trattava di realizzare un progetto ambizioso e strutturalmente complesso. Infatti mentre da un lato offre l'utilizzo di funzionalità avanzate e sofisticate, dall'altro le propone in modo intuitivo e facilmente fruibile, tanto da renderlo un ecosistema facilmente esportabile sia in contesto amatoriale che industriale. Un occhio di riguardo è stato rivolto alla sicurezza e alla scalabilità dell'architettura.

Per quanto riguarda il prossimo futuro, l'intenzione è quella di aggiornare l'applicazione lato server alla nuova versione di LoopBack interamente scritta in TypeScript e introdurre la possibilità di fornire applicazioni a pagamento.

Un ulteriore sviluppo sperimentale potrebbe essere l'implementazione della compra-vendita dei dati delle Thing tramite microtransazioni machine-to-

machine appoggiandosi su una criptovaluta di nuova generazione progettata specificamente per questo compito (es. IOTA[1]).

Il lavoro non deve essere considerato definitivamente concluso, infatti sarà necessario mantenerlo costantemente allineato con l'evoluzione dei documenti W3C. Infatti a tutt'oggi, (e presumibilmente anche in futuro) sono in corso modifiche che impattano anche in modo sostanziale su alcuni aspetti dello standard.

[1]https://www.iota.org/

Bibliografia

Articoli

[Mic13] Rodger Lea Michael Blackstock. ≪Toward Interoperability in a Web of Things≫. In: *UbiComp '13 Adjunct Proceedings of the 2013 ACM conference on Pervasive and ubiquitous computing adjunct publication* (2013), pp. 1565–1574.

[Som13] Abhijan Bhattacharyya Soma Bandyopadhyay. ≪Lightweight Internet protocols for web enablement of sensors using constrained gateway devices≫. In: *2013 International Conference on Computing, Networking and Communications (ICNC)* (2013), pp. 334–340.

[Pra15] Pramod Anantharam Pratikkumar Desai Amit Sheth. ≪Semantic Gateway as a Service architecture for IoT Interoperability≫. In: *2015 IEEE International Conference on Mobile Services* (2015), pp. 313–319.

[Rag15] Dave Raggett. ≪The Web of Things: Challenges and Opportunities≫. In: *Computer* 48.5 (2015), pp. 26–32.

[Sha15] Shanshan Zhao Shancang Li Li Da Xu. ≪The internet of things: a survey≫. In: *Information Systems Frontiers* (2015), pp. 243–259.

[LC19] Marco Di Felice Luca Sciullo Cristiano Aguzzi e Tullio Salmon Cinotti. ≪WOT STORE: Enabling Things and Applications Discovery for the W3C Web of Things≫. In: *Proceedings of IEEE CCNC 2019, Las Vegas, NV, January 11-14, 2019* (2019).

Libri

[Tri16] Dominique D. Guinard Vlad M. Trifa. *Building the Web of Things*. Manning Publications, 2016. ISBN: 9781617292682.

Online

[Ber06] Tim Berners-Lee. *Linked Data - Design Issues*. [ultima vista 25.02.2019]. 2006. URL: https://www.w3.org/DesignIssues/LinkedData.html.

[Gui11] Dominique Guinard. *A Web of Things Application Architecture – Integrating the Real-World into the Web*. [ultima vista 23.02.2019]. 2011. URL: http://www.vs.inf.ethz.ch/publ/papers/dguinard-awebof-2011.pdf.

[15] *F2F meeting: 20-22 April 2015 in Munich*. [ultima vista 23.02.2019]. 2015. URL: https://www.w3.org/WoT/IG/wiki/F2F_meeting:_20-22_April_2015_in_Munich.

[Zav15] Cosmas Zavazava. *ITU work on Internet of things*. [ultima vista 22.02.2019]. 2015. URL: http://wireless.ictp.it/school_2015/presentations/secondweek/ITU-WORK-ON-IOT.pdf.

[Eas17] Gary Eastwood. *IoT's interoperability challenge*. [ultima vista 23.02.2019]. 2017. URL: https://www.networkworld.com/article/3205207/internet-of-things/iots-interoperability-challenge.html.

[Kha17] Daniel Khan. *Understanding Garbage Collection and Hunting Memory Leaks in Node.js*. [ultima vista 27.02.2019]. 2017. URL: https://blog.codeship.com/understanding-garbage-collection-in-node-js/.

[17] *Participants in the Web of Things Working Group*. [ultima vista 23.02.2019]. 2017. URL: https://www.w3.org/2000/09/dbwg/details?group=95969&order=org&public=1#stats.

[Kel18] Gregg Kellogg. *W3C JSON-LD 1.1*. [ultima vista 25.02.2019]. 2018. URL: https://www.w3.org/TR/json-ld11/.

[W3C18] W3C. *Web of Things (WoT) Security and Privacy Considerations*. [ultima vista 24.02.2019]. 2018. URL: https://www.w3.org/TR/wot-security/.

[19a] *About W3C*. [ultima vista 23.02.2019]. 2019. URL: https://www.w3.org/Consortium/.

[Sta19a] Statista. *Internet of Things (IoT) connected devices installed base worldwide from 2015 to 2025 (in billions)*. [ultima vista 28.02.2019]. 2019. URL: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[Sta19b] Statista. *Size of the global Internet of Things (IoT) market from 2009 to 2019 (in billion U.S. dollars)*. [ultima vista 28.02.2019]. 2019. URL: https://www.statista.com/statistics/485136/global-internet-of-things-market-size/.

[W3C19a] W3C. *Web of Things (WoT) Architecture*. [ultima vista 22.02.2019]. 2019. URL: https://w3c.github.io/wot-architecture/.

[W3C19b] W3C. *Web of Things (WoT) Protocol Binding Templates*. [ultima vista 22.02.2019]. 2019. URL: https://w3c.github.io/wot-binding-templates/.

[W3C19c] W3C. *Web of Things (WoT) Scripting API*. [ultima vista 22.02.2019]. 2019. URL: https://w3c.github.io/wot-scripting-api/.

[W3C19d] W3C. *Web of Things (WoT) Thing Description*. [ultima vista 22.02.2019]. 2019. URL: https://w3c.github.io/wot-thing-description/.

[19b] *Web of Things Working Group*. [ultima vista 24.02.2019]. 2019. URL: https://www.w3.org/WoT/WG/.

[19c] *Web of Things Working Group Charter*. [ultima vista 24.02.2019]. 2019. URL: https://www.w3.org/2016/12/wot-wg-2016.html.