Alma Mater Studiorum · Università di Bologna

**SCUOLA DI SCIENZE**
**Corso di Laurea in Informatica**

# Studio ed implementazione di un protocol binding per dispositivi Tuya nel W3C Web of Things

**Relatore:**
**Chiar.mo Prof.**
**Marco Di Felice**

**Presentata da:**
**Alessio Pellegrino**

**Correlatore:**
**Dott.**
**Luca Sciullo**

**II Sessione**
**Anno Accademico 2020 / 2021**

# Abstract

In questa tesi triennale verrà discussa l'implementazione di un subprotocol HTTP nel progetto node - Wot [6] per l'utilizzo dei device IoT Tuya [34] tramite lo standard Web of Things definito dal consorzio W3C [39].
In particolare si è adattata l'interazione WoT alle API Tuya di modo che l'utilizzo risulti totalmente trasparente all'utente finale che avrà l'impressione di interagire con una Thing WoT classica. Il binding ha come obiettivo l'espansione dell'ecosistema WoT. Per fare ciò si vuole aggiungere ad esso una serie di dispositivi già pronti all'uso ed in vendita nella maggior parte dei negozi oltre che presenti nelle case delle persone. Verrà poi presentato un semplice caso d'uso per mostrare le possibilità del binding soprattutto legate alla *smart home*. Il binding permette la compatibilità con tutte le Thing Tuya ad oggi presenti sul mercato indipendentemente dal produttore e dalla loro natura oltre che aprire alla possibilità, in futuro, di integrare altre API messe a disposizione dai server Tuya.

# Indice

# Elenco delle figure

# Capitolo 1

# Introduzione

Grazie alla miniaturizzazione dei microprocessori e alla sempre più capillare diffusione della rete internet, dai primi anni 2000 si è iniziato a parlare di "*Internet of Things*" (spesso abbreviato in IoT). Per IoT si intende una serie di oggetti (Thing) che, una volta collegati alla rete internet, moltiplicano le loro potenzialità.
Se nei primi anni si è parlato solo di dispositivi dotati di RFID [16] passivi che aumentavano l'iterazione solo marginalmente oggi si può trovare in vendita a pochi euro ogni tipo di dispositivo per la domotica: dalle lampadine alle prese per la corrente passando per grandi e piccoli elettrodomestici, tutti che interagiscono con l'utente nel più disparato dei modi e con le più diverse tecnologie.

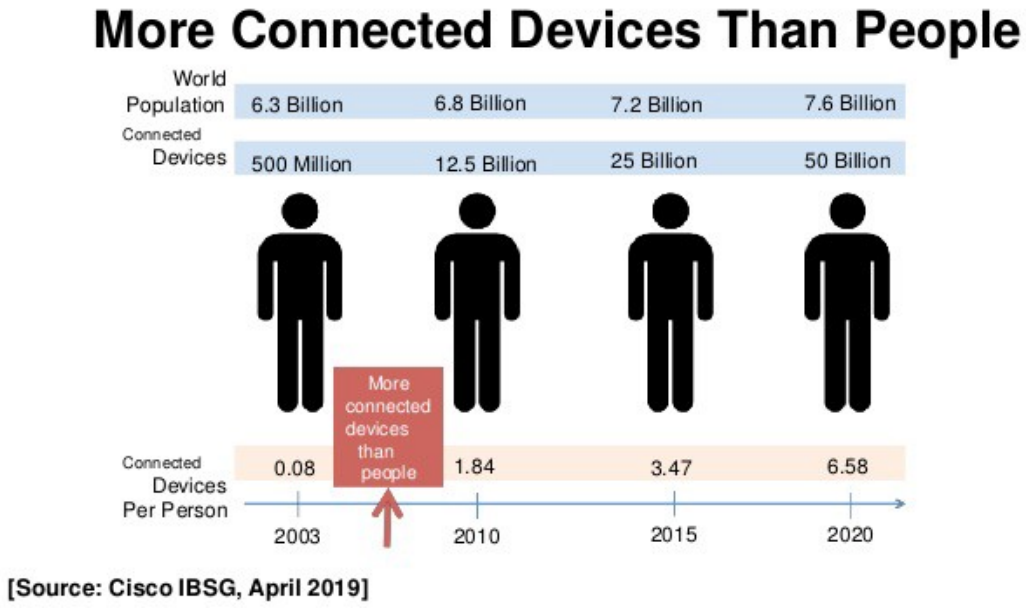


Figura 1.1: Numero di device connessi per persona

Oggi la diffusione di questi dispositivi non è un problema vista anche la rapida crescita dell'adozione degli stessi. Infatti in figura 1.1 è possibile vedere il numero di device ad oggi connessi (compresi cellulari e PC). L'immagine non da quindi un dato relativo solo al mondo dell'IoT ma fornisce comunque un'informazione indicativa della portata del fenomeno. Nonostante quindi i device IoT siano sempre più presenti nella vita delle persone, sorge il problema della frammentazione software dei dispositivi che faticano a comunicare tra loro quando non fanno capo allo stesso marchio. La causa di questo problema è da ricondurre principalmente a due fattori particolarmente impattanti: in primo luogo le differenti necessità di connessione dei dispositivi (un sensore di umidità deve consumare meno e inviare/-ricevere meno dati rispetto ad una lavatrice). In secondo luogo la poca disponibilità dei produttori a lasciar comunicare i loro dispositivi con i concorrenti per indurre l'utente a restare nell'ecosistema creato. Per questo, in ambito accademico, si è sviluppato il concetto di Web of Things: un nuovo modo di connettere i dispositivi al Web per sfruttare le caratteristiche di quest'ultimo e facilitare l'interoperabilità delle Thing. Nel 2015 il W3C ha imbastito un working group che si occupa di diffondere lo standard WoT definito dal W3C stesso. Il working group ha quindi creato un progetto open source [6] che implementa lo standard WoT. Questo progetto inserisce delle modifiche al node - WoT del working group che permette di sfruttare le Thing Tuya come fossero Thing WoT. Per fare questo è stato creato un sub protocol HTTP che riconosce le Thing Tuya e si occupa di convertire le chiamate alle API Tuya in modo che i dati ricevuti siano conformi a quelli richiesti dal servient WoT. Viceversa, ai servient WoT, è permesso inviare dati come previsto dal protocollo del W3C ed il binding si occuperà di eseguire la conversione per rendere la chiamata conforme a quella prevista dai server Tuya. In questo modo il panorama di Thing WoT si amplierà con una serie di dispositivi economici e già presenti sul mercato e nelle case delle persone. Ad oggi infatti Tuya dichiara di avere più di 300 mila device che si interfacciano con le loro API [34] e questo binding permette, da subito, di renderle compatibili con il protocollo del W3c ed utilizzabili direttamente da node - WoT. Inoltre, è stato definito un caso d'uso per l'integrazione delle Thing Tuya in una *smart home* WoT per esemplificare le possibilità del binding.
I capitoli successivi di questo elaborato saranno così strutturati:

- Il capitolo due si concentrerà sullo stato dell'arte, quindi si discuterà del mondo delle Thing connesse e del loro stato attuale per poi passare a dettagliare come il WoT si inserisce nel panorama attuale, i suoi obiettivi, lo standard definito dal W3C che ne implementa la filosofia ed il suo stato implementativo nella forma del progetto node - WoT.

- Il capitolo tre si concentrerà sull'oggetto di questa tesi e come esso si inserisce ed interagisce con il progetto che va ad ampliare.

- Il capitolo quattro discuterà gli aspetti implementativi del lavoro svolto e le problematiche legate all'interazione con i server Tuya.
- Il capitolo cinque illustrerà un semplice caso d'uso esplicativo del lavoro svolto assieme ad una serie di misurazioni utili a comprendere l'overhead aggiuntivo che l'applicazione porta rispetto alle librerie ufficiali.
- Il capitolo sei conclude l'elaborato dettagliando possibili sviluppi e limiti del progetto.

# Capitolo 2

# Stato dell'arte

## 2.1 Internet of Things

Per Internet of Things o IoT si intende un qualsiasi oggetto (Thing) capace di collegarsi alla rete Internet per espanderne le possibilità e capacità di comunicazione. Non è però necessario che una Thing nasca tale ma anzi è possibile rendere device "offline" e passivi delle Thing tramite l'uso di controllori come Raspberry e Arduino.
Se in un primo momento l'interazione si limitava a macchine passive dotate di un qualsiasi Tag (codice a barre, RFID, QR code etc) che permetteva un limitato scambio di informazioni, oggi la miniaturizzazione dei microprocessori, la loro crescente potenza e la facilità di connessione ad Internet hanno reso molto più facile lo sviluppo in ogni campo di questo concetto fino a farlo diventare uno dei campi di maggior sviluppo ed interesse. Infatti, come è possibile vedere in figura 2.1, la diffusione negli anni di questa tecnologia è cresciuta molto velocemente ed essa non accenna a fermarsi.

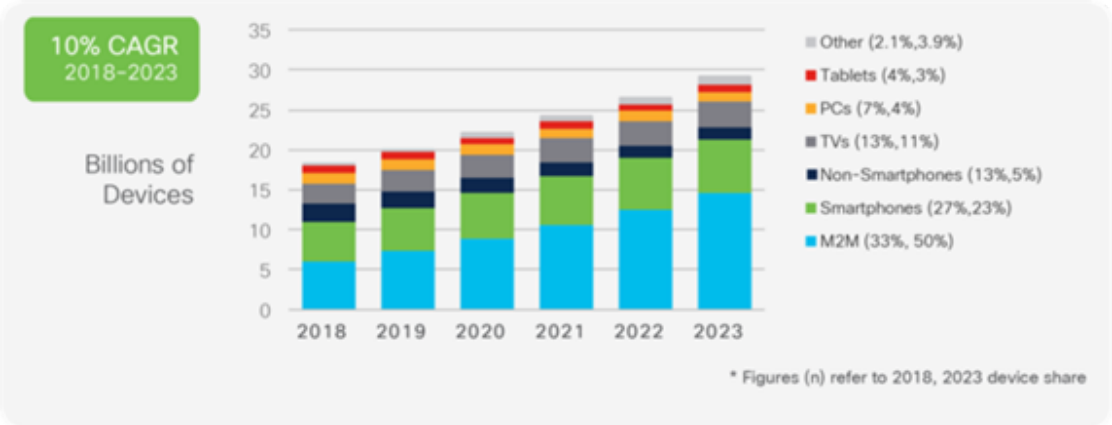


Figura 2.1: Dispositivi connessi divisi per tipologia

Il termine IoT è stato coniato da Kevin Ashton del MIT e presentato al P&G del 1999 [2]. Il concetto alla base ha l'idea che le persone, spostandosi in vari luoghi interagiscono con gli oggetti presenti [16]. Collegare alla rete Internet questi oggetti avrebbe migliorato l'interazione con essi e, di conseguenza, la vita delle persone. La naturale evoluzione di questo concetto ha poi portato le Thing sul Web. Difatti si comincia a parlare di Web of Things (al posto che di Internet of Things) già nel 2007 in un articolo di Erik Wilde, dall'università di UC Berkeley[40]. Oggi le Thing sono di natura molto eterogenea e spaziano quindi dalla semplice lampadina domestica fino ad arrivare ad automobili, macchine industriali ed interi edifici. La versatilità del Web permette, a fronte della sola necessità di un'interfaccia di rete, di poter soddisfare ogni differente esigenza e, contemporaneamente, dare un facile punto di accesso alla Thing sia ad utenti finali che ad eventuali software oltre che ad altre Thing connesse allo stesso modo. La sempre maggior capillarità della rete Internet non fa poi che espandere le potenzialità di questo concetto permettendo di poter ampliare in maniera facile e veloce l'ecosistema creato senza necessitare di ulteriori infrastrutture.

### Architettura IoT

Negli anni le vaste possibilità hanno portato alla creazione di diversissime applicazioni dell'idea di IoT: Thing di uso casalingo come lampadine, elettrodomestici o prese della corrente, usi medici, industriali (es: controlli remoti di stati di macchinari in funzione), edili ed agroalimentari. Nonostante la diversa natura dei vari dispositivi, essi condividono comunque la presenza di un microprocessore che controlla l'hardware. Esso è anche capace di elaborare i dati ottenuti ed è dotato di un'interfaccia di rete che gli permette di comunicare con l'esterno.

Date queste caratteristiche comuni la ricerca si è accordata sulla definizione di tre layer [19] principali che costituiscono l'architettura di una Thing IoT:

- **Livello percettivo** È il livello hardware che ottiene dati dall'ambiente circostante tramite l'uso di sensoristica presente sulla Thing. Questo layer è incaricato anche del processo di elaborazione dei dati che verranno poi condivisi tramite il layer di rete. Ovviamente in questo layer rientrano anche attuatori, per Thing che ne sono dotati (es cancelli smart). Quindi la comunicazione con i livelli superiori è duplice: infatti è possibile dare comandi alla Thing tramite un'apposita interfaccia e quindi comandi dal livello di rete possono cambiare lo stato o il funzionamento dell'hardware della Thing.

- **Livello rete** Si occupa della trasmissione di dati ad altri dispositivi connessi alla rete Internet. A seconda della tecnologia utilizzata si interfaccerà poi con diversi dispositivi intermedi come Router, Switch etc. Spesso i dati grezzi ottenuti dal livello percettivo sono molti ed è quindi compito di questo layer comprimerli in un formato facilmente condivisibile.

- **Livello applicativo** Livello che si occupa dell'autenticazione, l'integrità, l'accesso e la confidenzialità dei dati scambiati con la Thing.

È poi possibile definire un ulteriore layer, il layer di sicurezza che è però trasversale ai precedenti. Questo layer si occupa di fare in modo che tutti i dati ottenuti, elaborati e scambiati restino protetti e non accessibili senza autorizzazione, specialmente se i dati riguardano l'utente che usa la Thing.

### Metodi di integrazione

Per agevolare la comunicazione tra device spesso molti produttori hanno creato Hub o Gateway (che possono essi stessi essere Thing) che sfruttano i diversi protocolli per collegare tra loro Thing altrimenti isolate. Ma questo non è l'unico modello di integrazione possibile, difatti, come evidenzia il lavoro di Elkhodr et. al[7], ci sono tre possibili modelli di integrazione di una Thing:

- **Integrazione di rete** In questo modello di integrazione i sensori e oggetti Wireless comunicano con un unico Gateway / Hub chiamato *intermediario* che viene collegato da un lato alla rete locale interna e dall'altro alla rete Internet e che ha lo scopo quindi di gestione delle Thing e scambio dati.

- **Integrazione indipendente** In questo modello di integrazione ogni Thing è indipendente dalle altre con un proprio stack di rete ed un proprio indirizzo IP. Nonostante una delle principali sfide dell'IoT sia dotare ogni Thing di un proprio indirizzo indipendente, non è sempre conveniente dare ad ogni

dispositivo un proprio IP, specialmente per dispositivi molto economici e semplici. È inoltre importante puntualizzare che, nonostante l'indipendenza delle Thing sia d'aiuto allo scopo, non è abbastanza per creare una rete che permetta l'interscambio di informazioni tra Thing. Essa è infatti dipendente anche dall'interoperabilità delle Thing tra loro.

- **Integrazione ibrida** Questo modello di integrazione fonde le due precedenti e si pone come obiettivo quello di farle coesistere. La necessità di questa integrazione nasce dal fatto che non è sempre conveniente sfruttare un unico modello d'integrazione e, spesso, conviene usare il metodo più adatto alle esigenze del progetto. Dando quindi flessibilità alla rete è possibile integrare le Thing tra loro senza dover forzare l'adozione di uno stack di rete univoco che potrebbe essere sconveniente.

### 2.1.1 Casi d'uso IoT

Data l'immensa versatilità del concetto di IoT è possibile sfruttarlo in diversi modi per aumentare l'efficienza e la semplicità di utilizzo di praticamente ogni device. L'uso di sensori ed attuatori che comunicano tra loro può permettere di creare analisi statistiche, raccogliere dati o creare scenari automatizzati che reagiscono sotto certe condizioni. Nonostante le applicazioni siano innumerevoli, è possibile distinguere delle macro aree di influenza: (i) trasporti, (ii) veicoli, (iii) industria, (iv) sanità, (v) agricoltura, (vi) casa smart, (vii) città smart, (viii) uso commerciale.

#### Trasporti e veicoli

Automobili, camion, aereo-mobili e navi sono già dotati di un elevato numero di sensori, dare ad essi una maggiore possibilità di connessione potrebbe semplificare aspetti chiave del veicolo come la manutenzione o la gestione degli stessi. Altri mezzi di trasporto invece, se dotati di sensori, possono espandere di molto le loro possibilità come nel caso delle biciclette che, grazie alla possibilità di essere localizzate, hanno creato un nuovo mercato di bike sharing che ha migliorato la mobilità in molte città[29]. Inoltre è possibile espandere ancora di più le possibilità di veicoli come camion o automobili usando sensori per monitorare lo stato dei beni trasportati, aumentare l'interconnessione tra veicoli per gestire il traffico, migliorare la gestione del viaggio tramite mappe interattive. Le possibilità del trasporto smart[43] hanno radicalmente cambiato il modo di intendere il trasporto e, grazie ai progressi che compagnie come Alphabet o Tesla stanno facendo nel campo del self driving la ricerca in questo campo si sta ampliando ed è decisamente importante.

### Industria

L'obiettivo in questo campo è di dotare ogni macchinario industriale di sensori per la diagnostica di problemi, l'analisi dei processi produttivi e l'ottimizzazione della filiera produttiva[41]. Inoltre si possono analizzare i prodotti per verificare non ci siano imperfezioni o dotarli di tag che, se associati al GPS di un'autovettura che li trasporti, ne monitorino i tempi di spedizione una volta usciti dalla produzione.

### Smart Health

L'uso dei dispositivi IoT nella medicina può portare molti benefici[17]. I principali apporti di questo tipo di tecnologia possono essere divisi in 4 macro aree identificate da [3]:

- *Tracking*: la possibilità di seguire e monitorare il paziente in maniera più immediata è sicuramente uno dei più grandi benefici di questa tecnologia. Allo stesso modo possono poi essere identificati e monitorati gli strumenti usati per verificarne lo stato di deterioramento.

- *Identificazione ed autenticazione*: migliorare il processo di identificazione dei problemi del paziente non solo aiuta l'operatore medico con una diagnosi più semplice ma assicura al paziente migliori cure e limita il rischio di errori umani che possano danneggiarlo.

- *Raccolta dati*: l'automatizzazione di raccolta dati assieme ad un sistema di condivisione di dati sanitari (tramite RFID) può velocizzare il processo di conoscenza del paziente e la sua, eventuale, ospedalizzazione.

- *Rilevamento*: sensori estremamente miniaturizzati possono, se inseriti all'interno del corpo del paziente, monitorarne i dati h24 ed inviare i risultati a dispositivi dell'utente come smartphone o tablet. Il tutto ovviamente senza la necessità di interventi invasivi per il paziente.

### Smart city

Anche le città possono guadagnare dall'integrazione di device IoT:

- *Integrità strutturale degli edifici*: soprattutto per edifici storici o di interesse pubblico è importante tener monitorata la qualità della struttura. Questo può essere particolarmente vero in aree ad alto rischio sismico dove frequenti scosse potrebbero compromettere la preservazione di edifici e causare danni a persone ed oggetti. Monitorare i dati importanti tramite sensori connessi può dare una mano in questo senso oltre che controllare altri fattori importanti

che possono inficiare la salute di chi è presente nell'edificio come presenza di fumo e qualità dell'aria.

- *Gestione dei rifiuti*: le città possono essere dotate di raccoglitori della spazzatura connessi che inviino segnali quando pieni per ottimizzare la raccolta dei rifiuti.
- *Monitoraggio*: vari altri aspetti della città possono essere monitorati come il livello di inquinamento dell'aria, l'inquinamento acustico ed in generale ogni possibile informazione che possa aiutare i *decision maker* a migliorare la gestione della città o informare i cittadini sulle condizioni della stessa.
- *illuminazione intelligente*: un problema che, assieme a quello del cambiamento climatico, viene sempre più discusso è quello dell'inquinamento luminoso. Un sistema di illuminazione intelligente che faccia risparmiare energia e spenga le luci quando non necessario (strada vuota) può migliorare la qualità della vita dei cittadini residenti nella zona e far risparmiare.
- *consumi cittadini*: sempre legato al tema del limitare gli sprechi di risorse si possono dotare i principali consumatori di energia elettrica (come autobus elettrici o luci) di sensori che permettano di ottimizzarne l'utilizzo. Si può inoltre monitorare lo stato delle tubature della città per evitare perdite[28].

### Abitazioni smart

Sensori e attuatori possono automatizzare alcuni lavori ripetitivi per la casa che altrimenti dovrebbero essere fatti a mano da chi la abita. Inoltre sistemi di gestione intelligente della corrente potrebbero far risparmiare le persone attivando grandi elettrodomestici solo nelle ore in cui il costo della corrente è più basso o attivando i caloriferi di modo da tenere una temperatura costante per la casa e non consumare troppo. Parecchie aziende come Amazon, Google ed Apple stanno investendo sempre di più nel rendere smart la casa dei propri clienti ed anche se i costi sono ancora alti si stanno abbassando velocemente. Inoltre molti amatori usano microcontrollori come Arduino o Raspberry per creare device IoT in autonomia.

### Agricoltura

Molti dei processi ripetitivi dell'agricoltura possono essere automatizzati e resi più efficienti [27]. Possono essere utilizzati diversi sensori per il monitoraggio della salute della pianta, si possono usare sistemi per l'irrigamento automatico, droni e videocamere per evitare che le coltivazioni vengano compromesse. Inoltre, tramite la raccolta dati, si possono identificare nuove metodologie di coltivazione che ottimizzino spazi e sprechi.

### 2.1.2 Servizi tramite API

Molte Thing IoT usano servizi basati su API REST per implementare la comunicazione tra le Thing e i client che vogliono interagire con esse. Questi servizi, di solito, non hanno un interfaccia Web e quindi, pur implementando i classici protocolli del Web, le Thing che li sfruttano non possono considerarsi appartenenti alle Thing WoT.

#### API REST

Il concetto di API è stato applicato al Web da Roy Fielding[8] che ha, nel 2000 per la prima volta, proposto il modello REST (REpresentational State Transfer) o RESTful. Questo modello architetturale prevede che, alla richiesta di una risorsa, ne venga inviato un suo *stato rappresentativo*: l'informazione può essere rappresentata in uno dei formati HTTP[36] (JSON, XML, plain text, ...). Questo permette di evolvere il servizio e cambiare come la risorsa venga elaborata, gestita o salvata senza che il richiedente ne debba essere a conoscenza dato che l'interfaccia rimane la stessa. Un servizio che voglia implementare API RESTful deve soddisfare le seguenti caratteristiche:

- L'architettura deve essere composta da client, server e risorse con richieste gestite dal protocollo HTTP[36]. Separare la parte di interfaccia utente da quella di gestione delle risorse rende più facile portare l'interfaccia utente su diversi dispositivi aumentandone così la scalabilità.

- La comunicazione deve essere *stateless* e viene quindi richiesto che essa non preveda memorizzazione di informazioni del client da parte del server. La richiesta fatta al server deve contenere già tutte le informazioni necessarie all'elaborazione ed è quindi compito del client gestire il salvataggio della sessione. Questo aiuta con lo sviluppo del server che non deve mai preoccuparsi di gestire eventuali chiamate precedenti migliorando il processo di identificazione di errori, facilitando la scalabilità e rendendo più facile il ripristino del servizio nel caso di guasti. Di contro questo può comportare che la dimensione della comunicazione aumenti velocemente per funzioni che richiedano una storia di dati per funzionare.

- Le informazioni, previa autorizzazione del server, devono poter essere salvate in cache. Questo porta a diminuire la richiesta di risorse che cambiano poco nel tempo (es: una pagina web che descrive la rivoluzione francese) e che quindi può essere utile salvare localmente per evitare un'ulteriore richiesta nel caso fosse di nuovo necessaria.

- Un'interfaccia uniforme per i componenti e un formato di trasmissione standard delle informazioni. Questo rende più semplice la comunicazione e lo sviluppo ma, di contro, l'efficienza ne risente dato che usare formati standard invece di un formato ottimizzato per la risorsa scelta appesantisce il lavoro.

- Le risorse devono essere:

  - Identificabili e separate dalla rappresentazione inviata al client.
  - Manipolabili dal client, questo comporta che la rappresentazione deve avere in sè anche le informazioni per la sua manipolazione.
  - Ipermediali ed ipertestuali e, di conseguenza, un client deve poter, attraverso il link, individuare tutte le altre azioni disponibili.
  - Identificate tramite un URI.

- Deve essere presente un sistema di server gerarchici e trasparenti al client. Questo permette di migliorare la scalabilità, rende più facile la gestione di legacy client incapsulando i servizi legacy e permette di spostare risorse molto richieste ad intermediari comuni. Di contro questo sistema aggiunge overhead al servizio e può portare anche a pesanti rallentamenti, per questo è importante avere un sistema di caching delle risorse che controbilanci l'overhead aggiunto.

Per risorsa non si intende poi semplicemente un dato che può essere ottenuto ma si considerano risorse anche funzioni che elaborino dati o manipolino lo stato di un'informazione. Tutto quello che può essere raggiunto tramite un link ipertestuale è quindi una possibile risorsa. Questo modello si è poi diffuso sul Web ed è oggi usato anche per lo sviluppo di applicazioni anche al di fuori del Web.

### API Tuya

Tuya è principalmente un'azienda che vende software per la creazione di Thing IoT. Per fare questo ha messo a disposizione una piattaforma che permetta ai produttori hardware di implementare la parte smart delle loro Thing in maniera facile e veloce. Quindi molti piccoli produttori si affidano a Tuya per avere un servizio già pronto da sfruttare per le loro applicazioni [34]. Tuya fornisce una serie di API REST che permettano di interagire con le loro Thing. L'interazione con esse però non passa da un'interfaccia Web ma usa applicazioni mobile sviluppate per Android ed iOS.

## 2.2 Interoperabilità

Una parte importante dei vantaggi dell'IoT deriva dal fatto che le Thing possano comunicare tra loro per espandere le loro capacità e potenzialità. L'interoperabilità si occupa proprio di questo. L'interoperabilità dei device IoT può essere classificata in queste quattro tipologie[35][31]:

- **interoperabilità tecnica** I device comunicano a livello hardware tramite protocolli ed infrastrutture che rendono possibile una comunicazione *macchina a macchina*
- **interoperabilità sintattica** I device si scambiano informazioni tramite l'uso di particolari formati di file come JSON o XML. Lo scambio viene effettuato sfruttando i molti protocolli già definiti per lo scambio di queste tipologie di file
- **interoperabilità semantica** I device condividono la stessa interpretazione del significato dei dati scambiati
- **interoperabilità organizzativa** Si basa sul concetto di scambio di informazioni significative tra diverse organizzazioni. Non è un'interoperabilità semplice da ottenere perchè richiede diverse infrastrutture e sistemi informativi oltre che la coesione di interoperabilità semantica, tecnica e sintattica.

Negli anni la ricerca si è molto concentrata sui metodi migliori per ottenere l'interoperabilità nel mondo dell'IoT. L'interesse non è solo accademico data la presenza di diversi progetti europei che vanno in questa direzione (Arrowhead, Big IoT e Wise-IoT per citarne alcuni).

## 2.3 WoT

Il concetto di Web of Things nasce nel 2000 quando Kindberg inizia a sperimentare degli scenari in cui gli oggetti fossero raggiungibili dal Web[16]. In questo paper viene introdotto il concetto di *web presence*: Una "*home page*" per oggetti che definisce gli entry point per essi e ne dà una loro rappresentazione virtuale. Per farlo il paper introduce la possibilità di aggiungere dei server agli oggetti di uso comune come stampanti, proiettori o lavagne e di lasciare al server interno all'oggetto il compito di gestire la *web presence* di esso. Viene poi proposto di rendere raggiungibili le pagine di questi oggetti tramite URL che possano essere condivisi agli utenti tramite tecnologie di beaconing. Gli URL erano poi anche entry point per le funzioni delle Thing appena definite. Il concetto di uso di URI per l'indirizzamento al device viene poi espanso nel 2007 da Wilde[40] che immagina di poter dotare ogni

oggetto connesso di un URI e di rappresentarne un suo modello virtuale tramite formati dati come HTML, XML, JSON ottenibili tramite le API REST e l'uso del protocollo HTTP, sfruttando le chiamate GET, POST, DELETE, PUT. L'idea del Web of Things evolve e verrà definito da Guinard e Trifa come *The Web of Things is a refinement of the Internet of Things by integrating smart Thing not only into the Internet (network), but into the Web Architecture (application)*[9]. Gli stessi autori[11] propongono poi di sfruttare i concetti delle RESTful API per realizzare il Web of Things: piuttosto che usare il Web solo come protocollo di trasporto (come viene fatto oggi dalla maggior parte dei servizi WS-*)[10] si integrano le Thing nel Web come parte integrante e le si rende accessibili a tutti. Per integrare le Thing poi si propongono due metodologie:

- *Diretto*: dotando i device di Web server che possano renderli automaticamente parte del Web con un loro indirizzo IP e delle API che vengano gestite internamente alla Thing.

- *Indiretto*: collegando i device (tramite protocolli anche proprietari) ad intermediari (come Gateway o Hub) che si occupino della rappresentazione delle Thing sul Web. Questo concetto viene poi anche ripreso da [15] che immagina, per un'applicazione dell'IoT nel campo dell'agricoltura. Però il web, nella casistica immaginata nel paper appena citato, è "*solo*" usato come metodo per costruire Gui per l'IoT e non c'è quindi un'integrazione totale come immaginato in altri documenti.

Il WoT ha quindi come obiettivo quello di risolvere i problemi di interoperabilità del mondo degli oggetti connessi. Il termine si contrappone a quello che, oggi, si definisce *Web of People*[4]. Si immagina quindi un mondo dove i principali fruitori del Web non siano gli umani ma gli oggetti che ne sfruttano altri per aumentare la conoscenza dell'ambiente che hanno. Il Web of Things, piuttosto che discostarsi dal "*mondo reale*" lo abbraccerà rendendo gli oggetti che lo abitano capaci di *percepire* l'ambiente circostante ed usare informazioni condivise per reagire ad esso e quindi migliorare le vita di chi, il mondo, lo abita. Ad oggi, però, non esiste una definizione standard di Web of Things [32]. Questo può portare a confondere il concetto di IoT con quello di WoT. La differenza tra i due è che, nel primo caso, si parla degli oggetti che vengono connessi mentre, nel secondo, di una metodologia per la connessione degli stessi.

### 2.3.1 Applicazione di mashup

Un'applicazione di mashup è un'applicazione che, sfruttando i dati da diverse fonti li elabora per fornire nuove possibilità o nuove informazioni. Un possibile utilizzo di questo tipo di software che includa il mondo dell'IoT può essere un programma

che raccolga i dati di diversi sensori in giro per casa e li mostri in maniera agevole e con calcoli statistici a correlazione. È inoltre ovviamente possibile sfruttare applicazioni di mashup per attivare attuatori: se in casa si avesse un sensore di qualità dell'aria e lo si volesse collegare a delle finestre che si aprono e chiudono a seconda delle necessità si potrebbe implementare la logica del funzionamento tramite un'applicazione di mashup che legga i dati dal sensore di qualità dell'aria della stanza e apra le finestre una volta raggiunta una certa soglia critica. Aggiungendo poi un sensore di temperatura si potrebbe richiudere la finestra una volta che la stanza sia diventata troppo fredda.

### 2.3.2 Architettura del WoT

L'architettura di riferimento è quella proposta da Guinard et al. [12] che propone di integrare il WoT nell'architettura REST così da non dover reimplementare il sistema da basso livello. I device non sono obbligati ad avere un'interfaccia di rete ma possono appoggiarsi ad intermediari come proxie che gestiscano le chiamate REST al posto loro.
È quindi stato immaginato un sistema a layer illustrato in figura 2.2. Il primo layer ha lo scopo di rendere la Thing una Web Thing rendendola accessibile tramite i protocolli del web (http, Web socket etc) e usando API RESTful. Il secondo layer si occupa di dare la possibilità alla Thing di essere trovata ed utilizzata. Per fare ciò gli autori propongono di riutilizzare il concetto di web semantico[22] per descrivere le Thing e renderle trovabili sui tradizionali motori di ricerca. Il terzo layer si occupa di rendere i dati delle Thing condivisibili sul Web tramite l'utilizzo del socialWoT[5]. Questo layer si occupa anche della sicurezza e dell'efficienza della condivisione dei dati. È in questo layer che si parla infatti di autenticazione, protocolli di sicurezza (Oauth, Token etc) e di criptazione dei dati. Infine il quarto layer si occupa di costruire applicazioni di larga scala che sfruttino le Thing in modo utile ed intelligente per fornire strumenti e applicazioni.

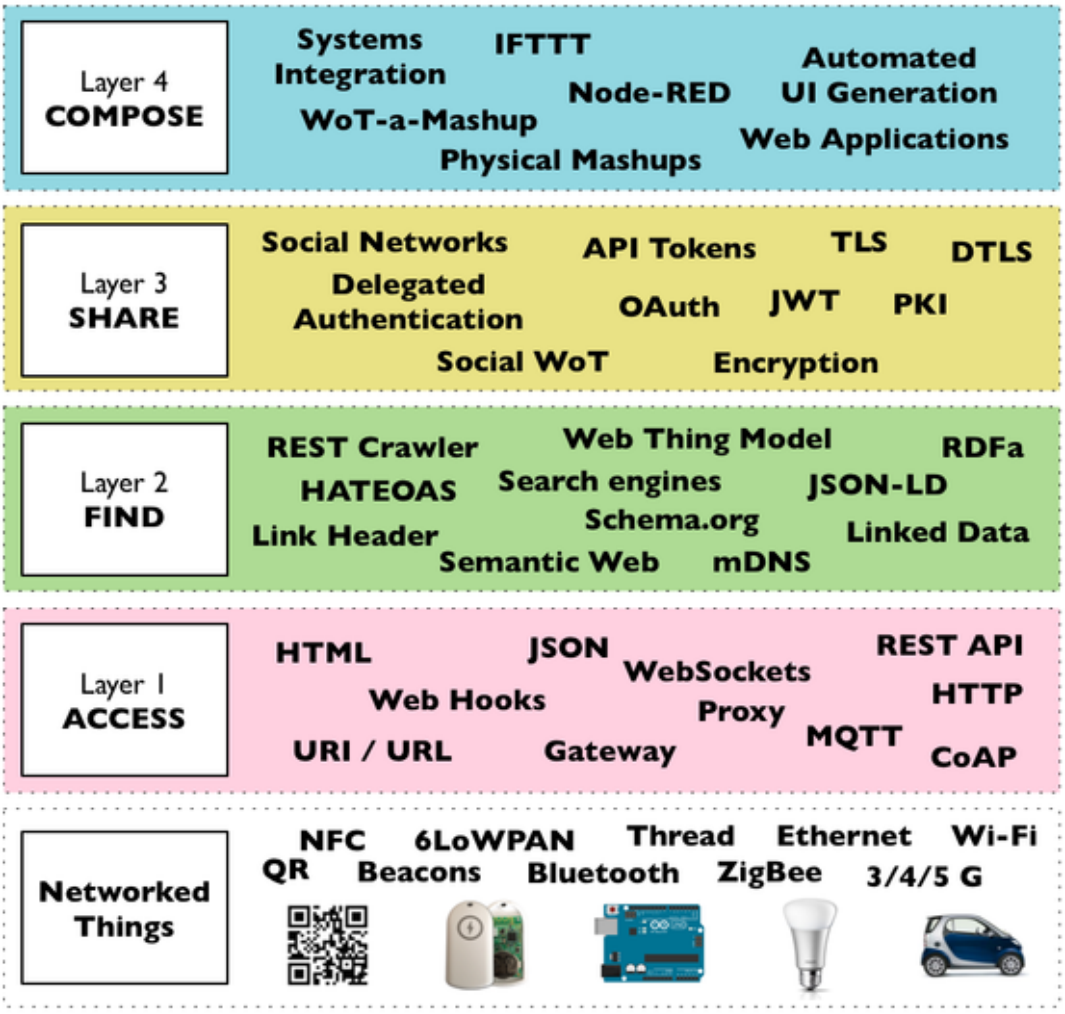


Figura 2.2: layer del Web of Things secondo il W3C

### Access

Come già discusso, Guinard et al.[11] propongono un modello di integrazione duplice a seconda della casistica sotto esame: per Thing più complesse e che possono consumare di più, si propone di integrare direttamente sulla Thing un web server che esponga un'interfaccia tramite API REST. In questo modo la Thing potrebbe gestire in autonomia la sua esposizione ed avere un proprio indirizzo di rete univoco. Questa integrazione prende il nome di *integrazione diretta*. Non tutte

le Thing possono però usare questa metodologia: device come sensori o piccoli attuatori non hanno abbastanza potenza di calcolo da permettersi di gestire un server e nemmeno hanno consumi energetici che possano permettere di gestire un carico simile. Per questo è possibile sfruttare l'integrazione indiretta che prevede ci sia un intermediario (solitamente gateway smart) che comunichi con le Thing tramite i loro protocolli proprietari ed espongano al posto loro un'interfaccia Web basata sull'architettura RESTful. Questi intermediari hanno quindi il compito di "tradurre" le richieste che ricevono al server ed inviarle alla Thing in un formato a lei accessibile. È inoltre possibile che un gateway esponga e gestisca diverse Thing di diversa natura. È poi compito di quest'ultimo astrarre sull'implementazione della comunicazione per esporre un'interfaccia uniforme che permetta lo sviluppo di applicazioni di mashup che usino servizi a livello user. Questa integrazione prende il nome di *integrazione indiretta*. Nel definire uno stack di layer per l'implementazione di un gateway per la trasformazione in Web Thing tramite *integrazione indiretta* Trifa et al.[33] hanno proposto un'architettura basata su tre layer: il primo, il layer di presentazione, ha il compito di rendere accessibili le Thing tramite API REST e gestire i diversi tipi di risorsa. Questo layer si occupa anche di dare ad ogni Thing un diverso URI. Il secondo layer, il layer di controllo, è composto da multipli plugin che vengono caricati per aggiungere funzionalità al gateway. I plugin possono essere molteplici ed arbitrari a seconda delle necessità. Due sono i plugin sempre presenti: il plugin di gestione dei device che si occupa di astrarre sulla gestione dei device ed il plugin di gestione degli eventi che comunica con la (o le) Thing e gestisce gli eventi una volta analizzati i dati ottenuti da esse. Il terzo layer, layer di astrazione, si occupa di creare un'interfaccia uniforme per utilizzare dispositivi anche con implementazione totalmente differente nello stesso modo.

### Find

Il metodo proposto per rendere le Thing trovabili sul Web è quello dello sfruttamento del Web semantico: l'idea è quella di dotare ogni Thing ed ogni funzione IoT di una descrizione semantica della loro utilità. Le Thing possono poi essere raggruppate per valori come provenienza geografica, unità di misura usata etc. I promotori di questa proposta[24] puntano quindi ad indicizzare la Thing nella ricerca dell'utente in base ai suoi parametri ed alle sue preferenze. Esiste poi la proposta di creare un motore di ricerca per il WoT: il WOT Semantic Search Engine (WOTS2E)[14] che avrebbe il compito di scandagliare il Web alla ricerca di endpoint per Thing WoT, distinguerle da quelle IoT ed indicizzare le Thing così ottenute in base alle loro descrizioni semantiche. Numerose altre proposte sono poi state fatte ([30] [23]) e la ricerca ancora prosegue.

#### Share

Per rendere pubblici i dati delle Thing si possono sfruttare diversi modi: si possono sfruttare le API REST per creare applicazioni dedicate. Un altro interessante approccio è quello del Social Web of Things[5] che propone di dare ai social network la possibilità di condividere i dati. Un simile approccio viene spesso usato dalla NASA (l'agenzia per l'esplorazione spaziale americana) per condividere i progressi delle sue missioni spaziali dotando i Rover inviati di una pagina Twitter dove espongono parte dei dati ottenuti [21]. È poi facile immaginare la possibilità per gli utenti di dare comandi alla Thing, magari tramite i sistemi di chat dei SNS.La Thing potrebbe poi rispondere a questi comandi con la medesima metodologia di comunicazione.

#### Compose

Questo layer dà la possibiltà di creare applicazioni di mashup che siano facili da costruire per dare la possibilità a più persone possibile di sviluppare software o prototipi di sorta. Allo stesso tempo è importante dare a chiunque la possibilità di interagire con le Thing e su questo si basa il lavoro di Mainetti et al. [20] che propone un sistema a 4 layer (accessibilità, esecuzione, proxy e composizione) per esporre dati con un software leggero e che funzioni su diversi tipi di hardware.

### 2.3.3 W3C - WoT

Il World Wide Web Consortium, W3C, è un'organizzazione non governativa che si occupa della definizione di standard per il Web. Esso si è posto il compito di diffondere lo standard WoT per facilitarne l'adozione da parte di produttori e sviluppatori. Per fare ciò è stato deciso di implementare l'astrazione necessaria dalle Thing usando paradigmi già esplorati in precedenza nel web per rendere più semplice e veloce lo sviluppo senza dover però sacrificare funzionalità ed efficienza. Di seguito verranno illustrate le caratteristiche del Web of Things allo stato dell'arte attuale [38]. La documentazione esposta è comunque ancora *work in progress* e quindi soggetta a cambiamenti successivi alla stesura di questo elaborato.

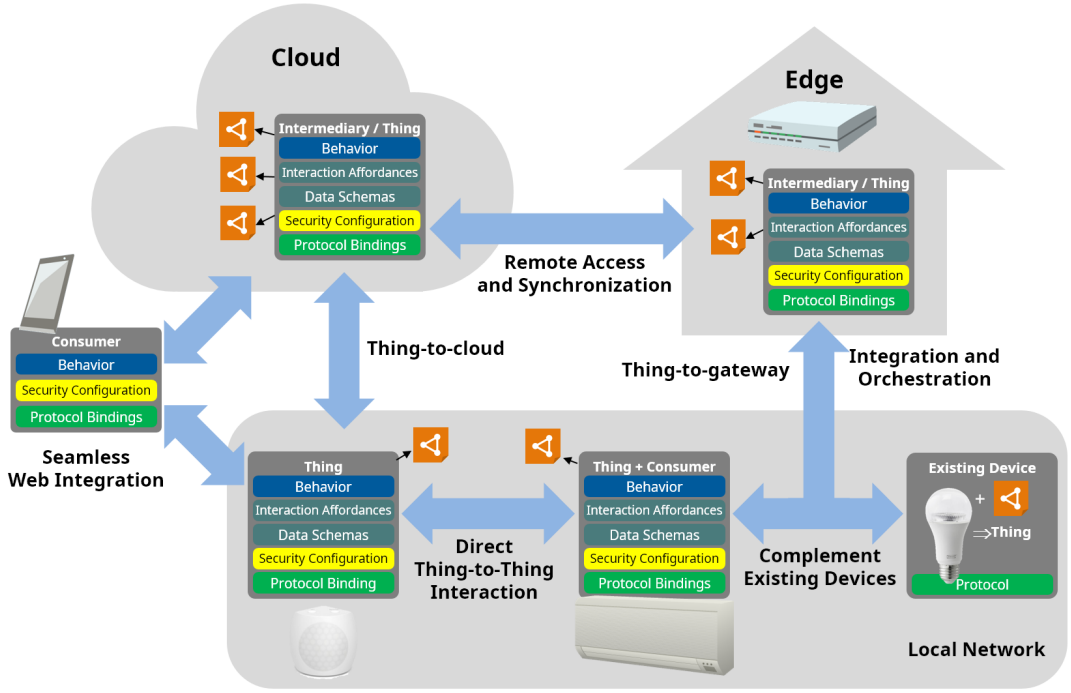


Figura 2.3: Architettura del Web of Things secondo il W3C

Le varie componenti del Web of Things vengono denominate Building blocks. Ogni building block ha il compito di rappresentare un aspetto architetturale della Thing esposta. I principali building blocks sono tre:

1. WoT Thing Description
2. WoT Binding Templates
3. WoT Scripting API

ai quali se ne può aggiungere un quarto, il layer di sicurezza, che è altrettanto fondamentale allo sviluppo del progetto.

### Architettura WoT - W3C

L'obiettivo del WoT - W3C è quello di mappare una Thing (fisica o virtuale che sia) tramite una TD, thing Description che ne descriva i dati e metadati, i metodi di interazione e di quindi astrarre dall'implementazione della stessa. Uno dei principali scopi del WoT è quindi quello di descrivere le caratteristiche di una Thing e come essa può interagire con il suo ambiente circostante. Essendo il concetto di

Thing totalmente arbitrario è possibile, nel WoT, mappare anche un insieme di Thing o astrarre su alcuni processi remoti di macchinari. L'astrazione del WoT - W3C permette quindi di ignorare gli aspetti implementativi e concentrarsi sulla descrizioni dei dati che si vogliono rendere accessibili. Per andare in contro alle esigenze di interconnessione tra Thing non è poi stato diviso il ruolo di client e server che può essere svolto anche dallo stesso device. Così facendo si rende possibile ad una thing consumarne un'altra permettendo così di creare un ecosistema che si espande a piacimento. Andiamo ora a dettagliare i building blocks che caratterizzano il W3C - WoT:

### WoT Thing Description

Può essere definito come "l'interfaccia" della Thing, contiene una rappresentazione della Thing per il suo consumo ed utilizzo. Tale metodo è direttamente suggerito dallo standard del W3C e nella sua realizzazione pratica sfrutta un file ti tipo JSON-LD. La TD rappresenta una struttura dati semantica e contiene informazioni sul contesto nel quale la Thing opera, vari informazioni testuali come nome ed ID, dati esposti, azioni ed eventi legati alla Thing stessa. Vengono inoltre specificati metodi di interazione e protocolli di sicurezza.

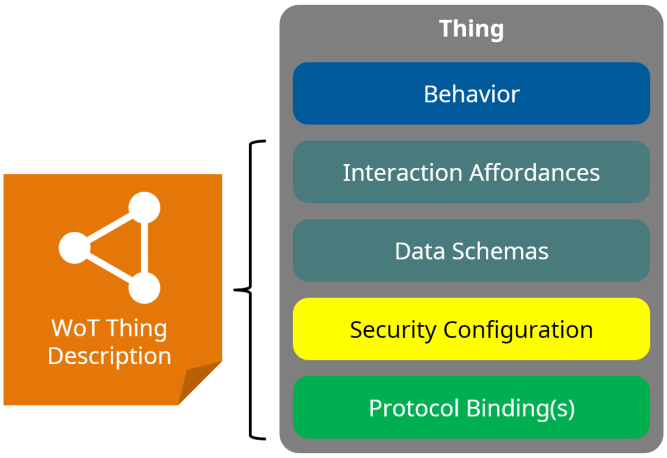


Figura 2.4: Schema del TD secondo il W3C

Le modalità di descrizione delle interazioni con la Thing vengono chiamate **Interaction Affordance**. Le Interaction Affordance vengono definite come *le proprietà fondamentali che determinano come un oggetto possa essere utilizzato* [37] e si dividono in 3 gruppi:

1. **Properties**
   Variabili che definiscono lo stato della Thing. Ogni property definisce il proprio tipo (booleano per lo stato di accensione di un lampadina, integer per le quantità di espresso rimanenti in una macchina del caffè, fino a tipi più complessi come oggetti e array) ed alcuni parametri aggiuntivi possibilmente utili come range e descrizione. Una property deve poter essere letta e può essere scritta (anche se non è obbligatorio lo sia). Nel progetto node - WoT, di cui si parlerà in seguito, è inoltre presente l'opzione di sola scrittura.

2. **Actions**
   Azioni che la Thing può compiere, possono modificare lo stato di più proprietà o innescare funzioni della Thing (es. far fare il caffè alla macchina del caffè). Da non confondere con la scrittura di properties che invece può solo modificare lo stato di una sola property. Ogni action specifica anche la tipologia di dati necessari da inviare nel caso ce ne fossero (oltre che eventuali informazioni utili come la descrizione).

3. **Events**
   Eventi che la Thing può scatenare sui quali un consumer può mettersi in ascolto (tramite le modalità specificate nel TD). Allo scattare della condizione espressa nell'evento (es. soglia di temperatura superata) la Thing invierà un segnale al (o ai) consumer in ascolto che potranno reagire nei modi da loro scelti.

Ogni Interaction Affordance ha poi come dato aggiuntivo un URI che ne definisce l'endpoint.

### WoT Binding templates

Come già discusso in precedenza ogni Thing ha, a seconda delle esigenze, un diverso protocollo di accesso ai dati. Sarebbe quindi controproducente forzare le Thing ad adottare un unico protocollo di comunicazione, per questa ragione il WoT non indica preferenze in tal senso ma pretende che il protocollo scelto venga indicato nel TD. Non è poi necessario che ogni Interaction Affordance segua lo stesso protocollo ma per ognuna di esse può essere specificato il metodo di interazione.

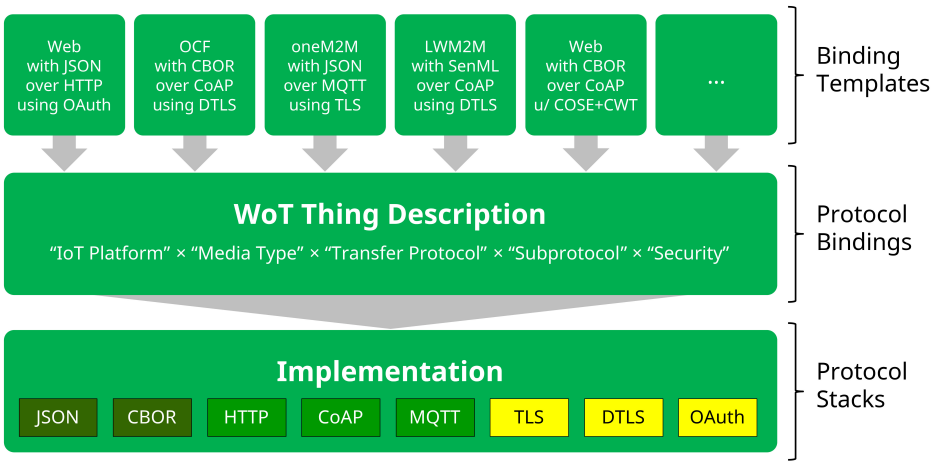


Figura 2.5: Binding templates secondo il W3C

Al consumer spetta quindi il compito, una volta conosciuto il Binding Template della Thing, di imbastire uno stack di comunicazione consono al protocollo necessario. Di seguito si elencheranno gli elementi che compongono il Binding Template:

- **IoT Platform** Si specificano eventuali modifiche proprietarie al layer di applicazione standard del protocollo. Es: header specifici necessari al funzionamento.
- **Media-Type** Indica il formato di rappresentazione dei dati necessario alla Thing per operare che può variare da Thing a Thing.
- **Transfer Protocol** È il protocollo utilizzato per lo scambio dati con la Thing, come già discusso non ne viene indicato uno specifico e sta alle necessità della Thing indicarne uno (HTTP, CoAP, ...).
- **Subprotocol** I protocolli possono supportare estensioni che, nel caso, devono essere dichiarate come tali. Un esempio è il longpoll per HTTP.
- **Security** Ogni Binding Template può supportare diversi protocolli di sicurezza come DTLS, OAuth etc.

### WoT Scripting API

È un building block opzionale che permette di definire API compliant alle specifiche ECMAScript. Sono implementate nella Thing e definiscono il suo comportamento o quello di un consumer o di un eventuale intermediario.

Queste API sono salvate in uno script caricato a runtime che quindi può essere riutilizzato da altre Thing e permette di abbattere i costi e aumentare la produttività. Le specifiche delle Scripting API dettagliano le interfacce di programmazione che permettono dalla produzione al consumo della Thing.

### Privacy e sicurezza nel protocollo WoT

Dato che la sicurezza e la privacy sono elementi trasversali che si applicano a tutti i building blocks il W3C non ha dato indicazioni specifiche ma solo linee guida da seguire nell'implementazione dello standard.

- **TD** Il TD deve essere disegnato in modo da limitare le informazioni date a client non autorizzati e non permettere la modifica di stati interni (senza autorizzazione).

- **Interfacce di rete WoT** Le interfacce di rete devono permettere le funzionalità essenziali evitando l'accesso non autorizzato e l'avvio di processi pesanti se non autorizzati.

- **Dati sensibili** Lo scambio di dati sensibili, soprattutto se riguardanti persone fisiche (es. la posizione) deve essere svolto solo tramite l'utilizzo di protocolli sicuri come l'HTTPS.

### WoT Servient

Il WoT servient è un componente software che incorpora tutti i building blocks e permette sia di consumare che di esporre Thing prendendo quindi il compito sia di client che di server per permettere facilmente alle Thing di interagire tra loro.

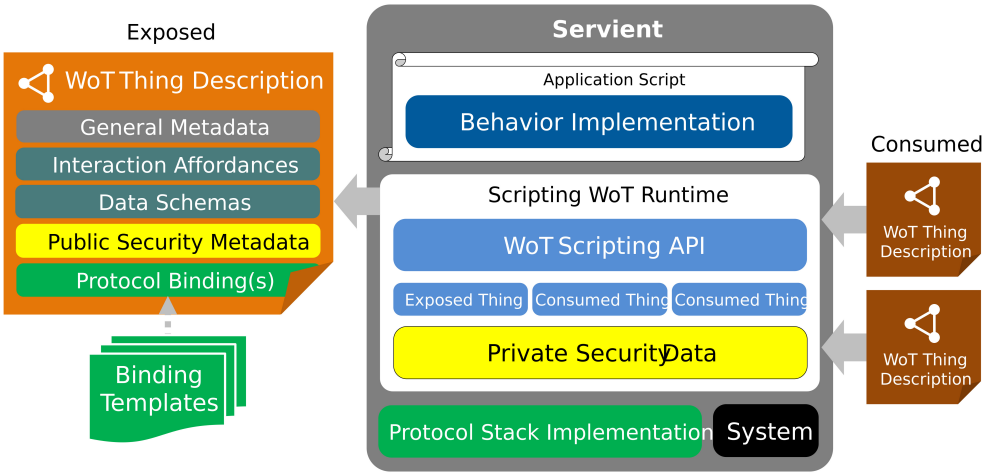


Figura 2.6: Architettura del servient WoT secondo il W3C

Di seguito verranno dettagliati i ruoli delle componenti che compongono l'architettura di un servient WoT come indicato nella figura 2.6:

- **System** Rappresenta l'hardware della (o delle dato che un servient può esporne più di una) Thing che il servient deve gestire e con cui deve comunicare. Lo standard non definisce nessuna metodologia di comunicazione tra il servient e l'hardware data la diversa natura delle possibili Thing. Inoltre può capitare che l'hardware e il servient non siano sullo stesso dispositivo ed è quindi compito di chi implementa la Thing mettere in comunicazione il servient e l'hardware usando i protocolli che più ritiene consoni.

- **Protocol Stack Implementation** Implementazione dei protocolli di comunicazione che realizzano l'interfaccia della Thing esposta. Il consumer poi utilizzerà l'interfaccia messa a disposizione per interagire con la Thing. La Thing esporrà anche metadati utili ad altre thing per facilitare l'intercomunicazione e, se possibile, i protocol stack produrranno messaggi di log specifici per la piattaforma e i protocolli utilizzati.

- **Scripting WoT Runtime** Il runtime gestisce tutta l'esecuzione della logica comportamentale della Thing esposta: processerà il TD, gestirà gli endpoint e caricherà le scripting API. L'ambiente runtime viene sempre eseguito in maniera protetta per evitare di esporre i dati sensibili che vengono gestiti dal sistema.

- **Behaviur Implementation** Descrive il comportamento e, di conseguenza, la logica comportamentale della Thing. Si compone di diversi script che defi-

niscono come vengono gestiti ed analizzati i dati in input / output, gestiscono le Interaction Affordance (che script collegare alla chiamata di un end point) e definiscono come comportarsi nel ruolo di consumer.

### Casi d'uso WoT

Il W3C ha infine indicato dei casi di utilizzo per le Thing WoT che verranno ora discusse:

- **Controller per Thing** La Thing viene controllata da un applicativo remoto che può essere un'applicazione dedicata ad esso o un browser e che venga usato da un utente per gestire la Thing. L'applicativo si comporterà come un client ed invierà comandi alla Thing che avrà il ruolo di server. Nel caso in cui invece la Thing supporti gli eventi le parti saranno invertite e sarà quindi la Thing a comportarsi da client per contattare l'applicativo che svolgerà il ruolo di server (es. un sensore di temperatura che raggiunge una soglia critica lancia un evento di surriscaldamento).

- **Thing to Thing** Le Thing comunicano direttamente tra loro, in questo caso è preferibile che le Thing usino entrambe uno stack di rete indipendente ma è comunque previsto che le Thing possano sfruttare un Hub per comunicare; come infatti è già stato discusso, è possibile, nel paradigma WoT - W3C, per un servient esporre una Thing non locale. Entrambe le Thing hanno in questo caso il ruolo di server ma almeno una delle due (o entrambe) deve avere il ruolo di client per interrogare l'altra.

- **Accesso remoto** È poi possibile configurare una Thing di modo che la si possa accedere anche al di fuori dalla rete locale. All'interno della rete locale non è necessario nessun meccanismo aggiuntivo di sicurezza perchè la rete è considerata sicura. Però è possibile collegare le Thing ad applicazioni per smartphone che cambiano spesso rete, sia WiFi che mobile ed in quel caso è quindi necessario configurare un layer di sicurezza aggiuntivo per la comunicazione con l'ambiente esterno.

- **Gateway** È un dispositivo utile alla connessione remota alla Thing: è collegato, tramite interfaccia di rete, sia alla rete locale che a quella Internet. Ha il ruolo sia di client che di server e reindirizza i comandi che ottiene da internet alle Thing a cui sono rivolti.

- **Digital twins** È una rappresentazione virtuale della Thing esposta che ne mima le possibilità. È gestito dal cloud o da un edge device, è utile a simulare la presenza della Thing nel caso non sia online la sua controparte fisica e lavora a stretto contatto con il gateway che comunica con la Thing fisica.

- **Cross-domain collaboration** Utile a mettere in comunicazione e collaborazione sistemi eterogenei di natura e dominio differente come città smart e abitazioni o industrie locate in posti distanti. Si possono avere due tipi di collaborazione cross-domain: *diretta* dove i due domini comunicano direttamente peer-to-peer ed *indiretta* dove due domini usano una piattaforma terza per scambiarsi informazioni. Per rendere più facile la collaborazione i sistemi devono esporre metadati sulle loro funzioni e possibilità.

- **Edge device** Simili a gateway ma con la capacità computazionale che gli permette di svolgere operazioni più complesse come operare come *bridge* tra vari protocolli di rete. Sono spesso utilizzati in contesti industriali dove pre - processano, filtrano e aggregano i dati.

- **Multi-cloud** La possibilità di caricare su diversi dispositivi nuove funzioni e caratteristiche della Thing esposta indipendentemente da versioni firmware presenti sugli stessi. Per fare questo si usa codice scritto per essere eseguito sul cloud e che quindi non implica che la Thing sappia implementarlo.

La figura 2.7 indica un'interazione esemplificativa tra i casi d'uso appena descritti.

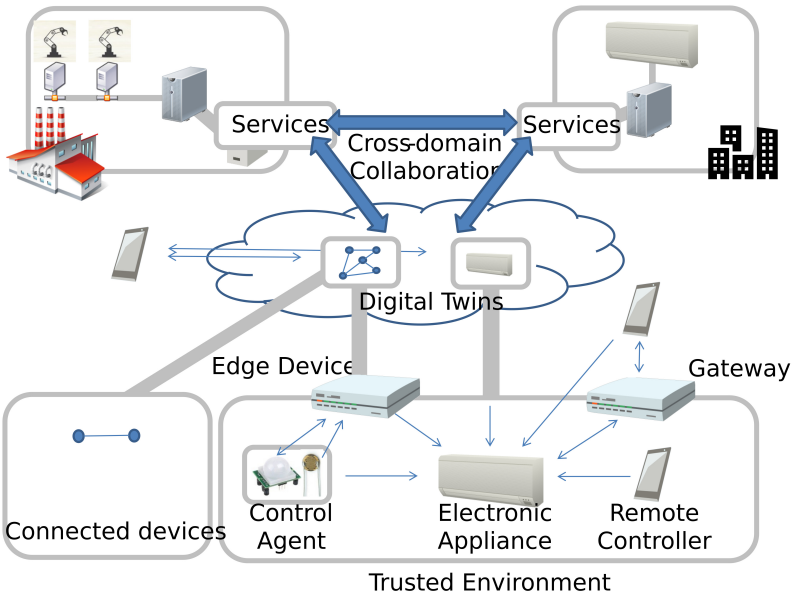


Figura 2.7: riassunto dei casi d'uso

### 2.3.4 Termini utilizzati

Nelle sezioni dedicate alla descrizione del WoT sono spesso stati utilizzati termini come ”consumer” ed ”exposer”. L'uso di questi termini non è stato casuale ed anzi sono quelli ufficialmente indicati dalla documentazione fornita dal W3C. Di seguito ne verranno illustrati i significati:

- **consumer** È il ruolo che assume il servient quando ”consuma” una Thing. Consumare una Thing significa fare il parsing della TD ed, eventualmente, sfruttarne le funzionalità.
- **exposer** È il ruolo che assume il servient quando ”espone” una Thing. Esporre una Thing significa rendere pubblico la sua TD e fare in modo sia possibile interagire con essa.

# Capitolo 3

# Studio ed implementazione di un protocol binding per dispositivi Tuya nel W3C Web of Things

In questo capitolo verrà trattato il progetto svolto, i suoi obiettivi e come esso si inserisce nel progetto del Working group del W3C. La descrizione sarà ad alto livello e non tratterà l'implementazione che sarà invece esposta in maniera più dettagliata ed approfondita nel capitolo 4 dedicato ad essa.

## 3.1 Obiettivi

L'obiettivo di questo elaborato è quello di espandere l'ecosistema delle Thing compatibili WoT aggiungendo ad esse le Thing compatibili Tuya [34]. Tuya è un'azienda cinese che produce una piattaforma per facilitare lo sviluppo di Thing IoT. Per questa ragione sono molti i produttori (piccoli e grandi) che si affidano ad essa per la creazione di prodotti smart. Ad oggi sono più di 5000 le aziende che collaborano con Tuya [34] e i prodotti basati sulla loro infrastruttura sono facili da reperire nei negozi. Pertanto aggiungere le Thing Tuya all'ecosistema WoT può aiutare la diffusione dello standard. Nonostante la piattaforma Tuya basi la comunicazione su un'iterfaccia composta da API REST, come è già stato visto succedere per molti servizi, l'infrastruttura Tuya usa i protocolli del Web solo come metodo di comunicazione e non ne sfrutta quindi le vere potenzialità. Ad oggi l'unico modo di interagire con una Thing Tuya è tramite un'applicazione disponibile per Android ed IOS e non esiste una pagina Web che ne permetta l'utilizzo. In alternativa è presente anche la possibilità di sfruttare le Thing tramite sistemi come quelli di Amazon Alexa e Google Home ma anch'essi non abbracciano i paradigmi del Web of Things come sono stati descritti nel capitolo 2. Il progetto

ha quindi come obiettivo quello di sfruttare le Thing Tuya, già largamente diffuse ed in continua crescita nelle case delle persone, come fossero Thing WoT facendo un binding delle API Tuya del tutto invisibile sia ad un utente finale sia ad un possibile programmatore di applicazioni di mashup. In questo modo, esponendo una Thing Description conforme alle specifiche del W3C e che mappi una Thing Tuya, è possibile consumarla esattamente come si consuma una Thing Description "*classica*" ed interagire con la Thing esattamente come si interagisce con una Thing WoT nativa. A tal fine è stato deciso di integrare le modifiche necessarie per il funzionamento del binding al progetto node - WoT proposto dal W3C working group. Questa scelta è stata fatta perchè il progetto è una completa trasposizione dello standard WoT del W3C. Di conseguenza il binding proposto può essere da subito operativo e permettere l'integrazione di tutte le Thing già presenti sul mercato.

## 3.2 Node - WoT

Il progetto node - Wot è diviso in componenti modulari che gestiscono le diverse parti caratterizzanti il consumo e l'esposizione della Thing.

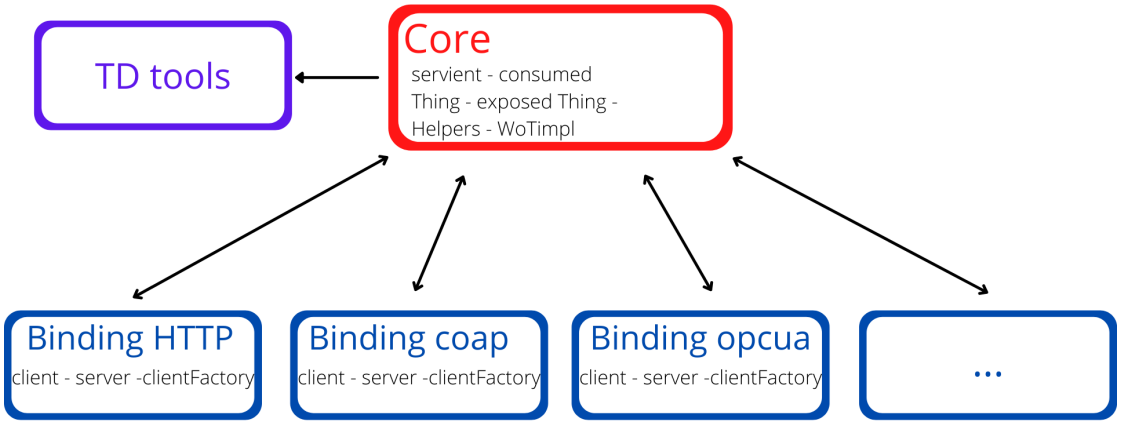


Figura 3.1: Struttura del node - WoT

- **Core**: è il componente che gestisce le Thing ad alto livello tramite vari sottocomponenti. Quando necessario vengono caricati a runtime i componenti del

progetto che si occupano a più basso livello della gestione della comunicazione tramite i protocolli più usati sul web.

  - Servient: è il sotto-componente che gestisce l'intera comunicazione e permette di consumare ed esporre Thing e aggiungere nuovi client o nuovi server. È il principale componente del node - WoT e tramite esso è possibile creare nuove Thing o applicazioni di mashup.
  - Consumed-thing: è il sotto-componente che gestisce una Thing consumata e permette di interagire con essa tramite l'uso di client adatti. È il componente necessario ad una Thing per interagire con altre o per un'applicazione di mashup per gestire le Thing collegate ad essa.
  - Exposed-Thing: è il sotto-componente che gestisce una Thing esposta e permette di definire come comunicare con essa. Permette anche di specificare le componenti personalizzate delle Interaction affordance. È il componente fondamentale per la creazione di una nuova Thing.
  - Helpers: definisce alcune funzioni utili più a basso livello come la fetch che permette di fare una richiesta ad uno specifico URI. Di solito questa funzione è usata per ottenere la Thing Description di una Thing esposta.
  - WoT-impl: è il componente che permette di trasformare un TD in una Thing esposta o consumata. Inoltre anch'esso ha alcune funzioni di supporto utili agli altri componenti del Core.

- td-tools: componente che contiene tutte le specifiche per la gestione e creazione delle Thing Desciption. È un componente che aiuta lo sviluppo di nuove Thing.

- binding-*: per poter comunicare con i vari protocolli adatti ad ogni Thing è stato scelto di creare dei binding che implementassero ognuno un protocollo diverso così da poter caricare runtime quello corretto. Ogni binding ha diversi componenti fondamentali:

  - server: componente che permette di creare un server per esporre Thing. Esistono diversi tipi di server (coap, fujitsu, http, mqtt, oracle, websockets) ed ognuno gestisce le Thing che espone secondo il protocollo che implementa. L'utilità di avere diversi tipi di server come componenti separate è quella di poter caricare di volta in volta il componente necessario senza dover cambiare gli altri.
  - client: come per i server anche in questo caso ci sono diversi tipi di client: coap, file, http, modbus, mqtt, netconf, opcua, websocket. Il client serve a comunicare con la Thing tramite il protocollo più adatto

ed il motivo per cui è un componente separato, come per il server, è per avere la possibilità a runtime di scegliere quale client caricare a seconda del protocollo che si intende usare.

– clientFactory: Ogni binding che implementi un client implementa anche una clientFactory. Esso fornisce i client alle componenti Core per l'interazione con le Thing tramite il corretto protocollo. I client possono sia essere creati ex novo oppure, nel caso fossero già presenti da precedenti utilizzi, essi possono essere recuperati dalla cache senza necessità di dover essere reinizializzati.

Oltre ad essere possibile caricare separatamente a runtime i diversi binding a seconda delle necessità è anche possibile, tramite *npm*, scaricare le dipendenze delle sole componenti che si intende utilizzare rendendo il progetto node - WoT configurabile a piacimento anche nello spazio di memoria su disco che esso occupa. Questo è particolarmente importante per alcune Thing dalle risorse limitate. Il lavoro di integrazione delle Thing Tuya sarà principalmente fatto andando a lavorare sulla sezione di client e server ed in particolare il client e server del *binding HTTP* creando così un *subprotocol* che gestisca le interazioni con i soli server Tuya. La necessità di sfruttare questo particolare binding nasce dal fatto che le API RESTful messe a disposizione da tuya sfruttano il protocollo HTTPS. Il binding opera quindi solo a livello client per gestire la comunicazione e l'exposer non comunica mai (se non eventualmente come client) con la Thing che espone.

### 3.2.1 Thing Description

Per ogni affordance interaction, node - WoT, predispone un form di possibile interazione disponibile. I form vengono impostati dal server che espone la Thing e sono personalizzati a seconda del protocollo utilizzato. In questo modo i client che devono creare la connessione con le affordance interaction della Thing possono impostare i parametri di connessione nella maniera richiesta dalla Thing.

## 3.3 Requisiti

Verranno ora discussi i requisiti tecnici e funzionali che delineano lo scope e le modalità di utilizzo del progetto.

### 3.3.1 Requisiti tecnici

Il progetto è stato considerato funzionante quando sono stati rispettati i seguenti requisiti:

- **Comunicazione con API Tuya** Il client HTTP di node - WoT deve potersi collegare alle API Tuya rispettando la costruzione del pacchetto HTTP da loro richiesta. E stato quindi necessario implementare l'ottenimento del token e la firma del pacchetto secondo le specifiche richieste dalle API.

- **Trasparenza** La comunicazione deve essere possibile senza che nulla cambi nell'implementazione della stessa rispetto alla comunicazione con una Thing WoT *classica*. Questo aspetto è particolarmente importante se si vuole rendere effettiva l'integrazione delle Thing Tuya nel paradigma WoT del node - WoT.

### 3.3.2 Requisiti funzionali

- **Creazione della Thing Description** L'utente deve specificare quali properties deve essere possibile visualizzare. Non è necessario che ogni property che la Thing tuya espone sia mappata dalla TD ma di quelle non mappate non sarà ovviamente possibile ottenere il valore. Le properties mappate devono avere lo stesso nome nel TD che viene scelto dalle API Tuya. Questo perchè il loro nome verrà utilizzato per identificarle all'interno dei dati inviati e ricevuti dalle API. Ogni property mappata deve poi essere descritta manualmente: in nessun modo il binding va a modificare quanto scritto nella TD che quindi deve rispettare gli standard del W3C. In particolare nel compilare come comunicare con la Thing i dati devono avere lo stesso formato di quelli che si vogliono far arrivare alla Thing Tuya. In nessun modo il binding va a cambiare il contenuto del dato ma solo come essi vengono inviati: Il nome della Thing scritto nella Thing Description deve poi coincidere con l'ID della Thing Tuya che si vuole mappare in quanto esso verrà poi sfruttato per eseguire le chiamate ai server Tuya. Nei campi legati alla sicurezza si deve specificare che la Thing usa la security Tuya usando la keyword *TuyaCredential*.

- **Servient - client** Il servient che vuole consumare la Thing come fosse un client ed interagire con essa, deve avere le credenziali necessarie ad autenticarsi al cloud Tuya comunicante con la Thing Tuya esposta. Inoltre, il client deve supportare il protocollo HTTPS necessario per comunicare con i server Tuya.

- **Servient - server** Il servient che vuole esporre la Thing deve configurare un server HTTP / HTTPS con parametro "tuyaUrl" corrispondente all'URL a cui le Thing Tuya si connettono. l'URL deve essere formato nel seguente modo: *https://openapiOP.tuyaRG.com/v1.0/devices/* dove *RG* corrisponde

ad uno dei seguenti valori: *cn*, *us*, *eu*, *in* a seconda della regione di provenienza della Thing ed *OP* corrisponde ai valori: *-weaz* per le Thing dell'Europa occidentale e *-ueaz* per le Thing dell'America orientale. Il binding è solo lato client, questo vuol dire che facendo un'operazione di lettura o scrittura sulle properties della *Exposed Thing* non sarà possibile leggere o scrivere effettivamente sui server Tuya. Per farlo, sarà obbligatorio consumare la Thing Description e, da lì, eseguire letture e scritture.

### Credenziali Tuya

Per ottenere le credenziali necessarie alle API per interagire con la Thing occorre eseguire i seguenti step:

1. Registrarsi alla piattaforma developer di Tuya al link *https://auth.tuya.com/*.

2. Creare un nuovo progetto Cloud.

3. Aggiungere i nuovi device nella sezione apposita. L'aggiunta deve essere fatta tramite la scansione di un QR code dall'app a cui le Thing Tuya sono collegate.

4. I dati necessari sono, nella schermata di riepilogo, l'*access ID* e l'*access secret*.

5. Aggiungere i dati necessari come mostrato nel codice sottostante dove il campo "baseUri" corrisponde all'URL a cui le API si devono connettere più la versione delle api utilizzate.

Purtroppo l'aggiunta di nuovi device non è supportata dal progetto node - WoT che si concentra solo sull'esposizione di nuove Thing e gestione di Thing già configurate. Le API Tuya danno ovviamente la possibilità di legare Thing ad un account e eliminarle dall'account da cui erano gestite ma l'implementazione di queste dinamiche non era nello scope di questo progetto.

```
"credentials":{
    "name":{
        "scheme":"TuyaCredential",
        "key":"key",
        "secret":"secret",
        "baseUri":"https://openapi.tuyaeu.com/v1.0"
    }
}
```

### 3.3.3 Compatibilità

La compatibilità del progetto è con la totalità delle Thing Tuya in commercio data l'omogeneità di funzionamento delle API. Nello specifico è poi stata testata con due differenti lampadine RGB prodotte da due diverse compagnie ed un sensore virtuale creato tramite il Cloud Tuya che mette a disposizione questa possibilità per fare test prima del deploy dell'applicazione. Il progetto si è occupato solo del funzionamento base delle Thing. Tuya mette però a disposizione diversi tipi di API (a pagamento o meno) da interrogare. Esse non sono per ora gestite ma non è difficile aggiungerle a quelle utilizzabili.


```
"title":"bff89e2b5a8e1516d6l89a",
"description":"an exemple TD of a real tuya smart rgb bulb. The id and region field are needed for connections porpouses",
"@context":["https://www.w3.org/2019/wot/td/v1"],
"securityDefinitions": {
    "TuyaCredential": {
        "scheme": "TuyaCredential"
    }
},
"security": [
    "TuyaCredential"
],
"properties":{
    "switch_led":{
        "type":"boolean",
        "description":"True if the bulb is on, false otherways"
    },
```


Figura 3.2: Esempio di parte di Thing Description Tuya prima di esporla

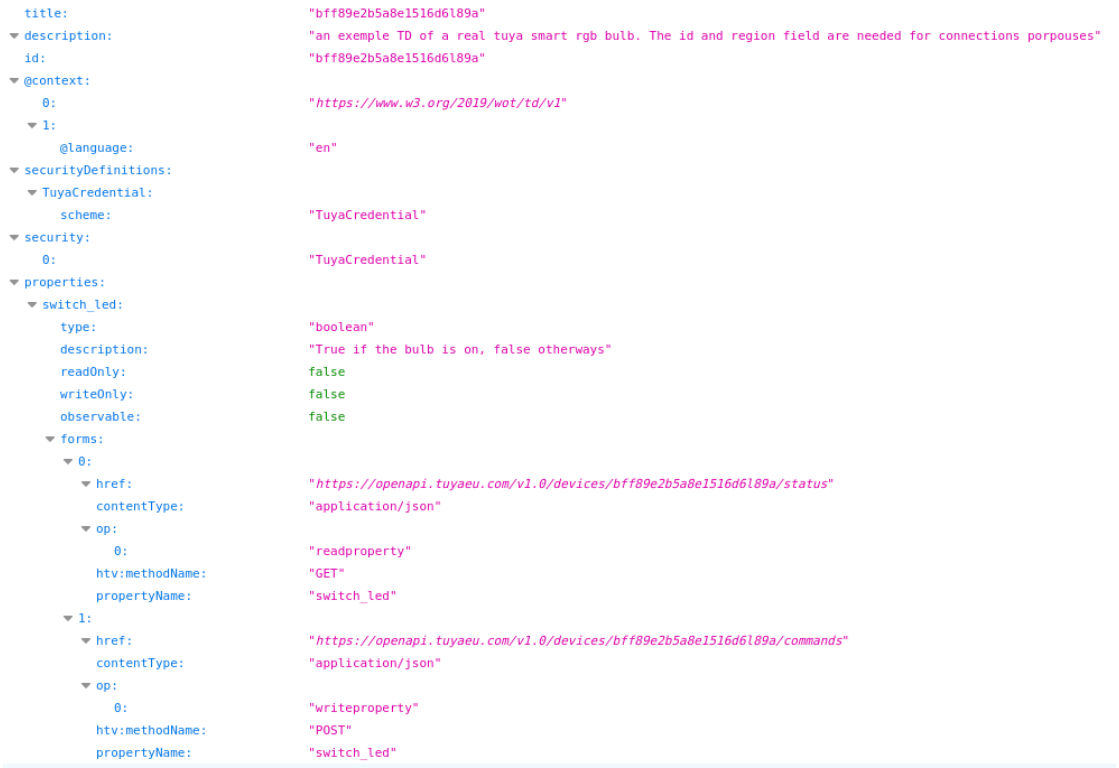


Figura 3.3: Esempio dello stesso TD dopo essere stato esposto

# Capitolo 4

# Implementazione

Il progetto è stato realizzato seguendo le tecnologie adottate dal progetto node - WoT dato che l'obiettivo è rendere quest'ultimo capace di consumare ed esporre Thing Tuya come spiegato nel capitolo 3. La repository del progetto è raggiungibile a questo link `https://github.com/UniBO-PRISMLab/binding-tuya`. Nel repository è stato caricato tutto il node - WoT compreso delle modifiche apportate dal binding. Questo capitolo sarà diviso in due sezioni: la prima, tratterà delle principali modifiche che il progetto ha apportato al node - WoT e come esse interagiranno con quest'ultimo. La seconda, più breve, dedicata alle tecnologie e librerie utilizzate per arrivare a questo scopo.

## 4.1 Implementazione

Il lavoro svolto si è concentrato su quattro componenti principali: Client HTTP, Server HTTP, le definizioni di sicurezza HTTP che il binding implementa e il file http che definisce alcuni tipi di form per le Thing.

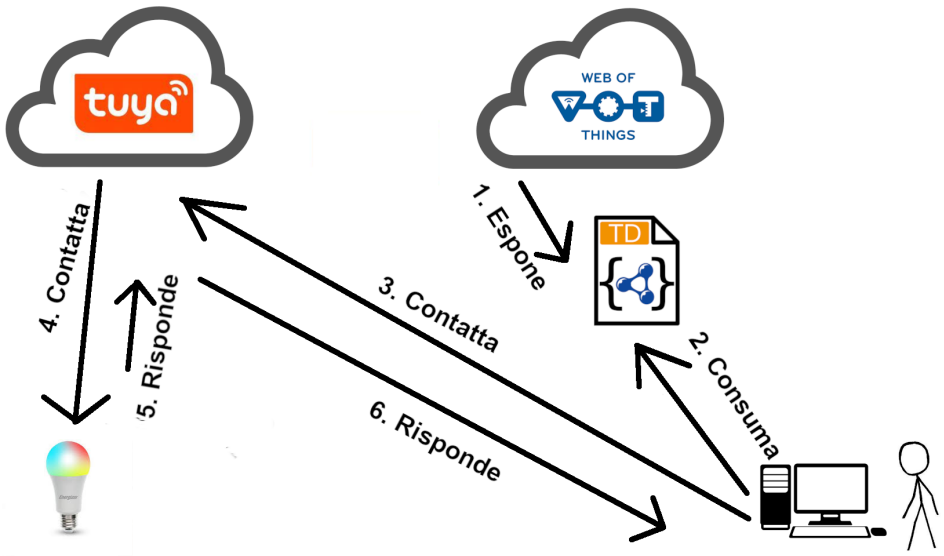


Figura 4.1: Esempio di interazione per le Thing Tuya

In figura 4.1 viene mostrato un esempio di come è stata implementata l'interazione tra le Thing Tuya ed il progetto node - WoT.

1. Un servient node - WoT tramite un server HTTP / HTTPS espone una TD di una Thing Tuya configurata come illustrato nel capitolo 3.3.2. Questo permette ad altri servient di consumare la TD e poter interagire con la Thing Tuya esposta.

2. Un consumer node - WoT consuma la TD della Thing Tuya. Per fare ciò ha bisogno di un client HTTP / HTTPS non necessariamente configurato per poter interagire con i server Tuya in quanto la consumazione del TD non necessita di credenziali Tuya.

3. Per interagire con la Thing consumata un client HTTPS contatta i server Tuya dopo aver riformattato i dati nel modo corretto. Il client in questo caso, invece, deve essere configurato con le impostazioni di sicurezza necessarie all'autenticazione ai server Tuya. Questo perchè l'interazione non passerà da chi espone la Thing Tuya ma avverrà direttamente con i server Tuya.

4. I server Tuya comunicano con la Thing ed eseguono la richiesta del client HTTPS. A questo punto il ruolo dell'exposer è compiuto e non entrerà più in gioco a meno che non si siano aggiunte action o eventi personalizzati gestiti da esso.

5. La Thing risponde ai server Tuya. Come avvenga la comunicazione tra server Tuya e Thing Tuya non è stato esplorato. Ci sono progetti open come [13] che tentano di comunicare direttamente con la Thing ma il lavoro è ancora incompleto.

6. I server Tuya inoltrano la risposta al client HTTPS.

7. Il client HTTPS riformatta la risposta in modo che sia conforme a quelle normalmente previste dalle Thing WoT. Per le writeProperty non sono previste risposte e quindi il client si limita a controllare che la chiamata sia andata a buon fine e scarta ogni informazione inviata dai server Tuya.

### 4.1.1 Client

Lato client sono state fatte modifiche alle chiamate riguardanti la Read e la Write Resource.

#### ReadResource

In questo caso nella funzione è stato aggiunto un controllo per riconoscere le chiamate dirette ai server Tuya. Nel caso si incontrasse una chiamata destinata ai server Tuya, dopo che questa sarà conclusa, il risultato verrebbe riformattato per restituire al chiamante esattamente il dato richiesto. Le API Tuya restituiscono sempre un array di oggetti. Ogni oggetto contiene due campi: un campo *"code"* che contiene il nome della property ed un campo *"value"* che contiene il valore della property (in formato stringified nel caso di oggetti, del tipo originale invece per ogni altro dato). Di fatto quindi ogni chiamata al server funziona come una "readAllProperties" e il dato corretto deve essere filtrato tramite lo scorrimento dell'array ottenuto dalla chiamata. Nel caso si stesse eseguendo una readAllproperties i dati verrebbero ricomposti con la giusta formattazione.

#### WriteResource

Anche in questo caso è stato aggiunto un controllo per riconoscere le chiamate dirette ai server Tuya. Nel caso si intercettasse una chiamata diretta ai server Tuya sarebbe necessario riformattare il body della chiamata. Le API Tuya prevedono infatti che ogni cambio delle proprietà venga fatto passando un array di oggetti denominato commands. Ogni oggetto contiene un campo *"code"* con il nome della property che si vuole modificare e un campo *"value"* con il nuovo valore della property. I dati al client vengono passati già trasformati in formato Buffer e questo comporta che debbano essere riconvertiti in formato plain text e poi di nuovo in Buffer. Questo implica un grosso overhead che è però impossibile da evitare senza

Listing 4.1: porzione di codice per il filtraggio del dato nella funzione readResource

```
if(form.href.includes("openapi.tuya")){
  let body = JSON.parse(await (buffer.toString()));
    if(!body.success) throw new Error(body.msg);
    let response: any;
    if((form as tuyaForm).propertyName == "all"){
      let total_response = [];
      for(let value in body.result){
        if(body.result[value].code == (form as tuyaForm).
            ↪ propertyName){
          total_response[body.result[value].code] = body.result[
              ↪ value].value;
        }
      }
      response = JSON.stringify(total_response);
    }else{
      for(let value in body.result){
        if(body.result[value].code == (form as tuyaForm).
            ↪ propertyName){
           response = body.result[value].value;
        }
    }
  }
  return { type: result.headers.get("content-type"), body: Buffer.
      ↪ from(response.toString()) };
}
```

fare modifiche alla parte core del node - WoT ed in particolare alla consumed Thing. L'overhead comunque, nonostante non sia nullo, è limitato dal fatto che tutte le properties richiedono dati che non occupano più di qualche carattere e quindi il loro encoding e decoding non prende più di una frazione di millisecondo. Nel caso si stesse eseguendo una writeAllProperties l'array non conterrebbe un solo elemento ma uno per ogni property presente nella Thing.

### 4.1.2 Server

Per la parte server si è lavorato soprattutto nella funzione di esposizione della Thing: è possibile configurare un server per esporre le affordance interaction ad un particolare URL (che verrà usato come base a cui aggiungere poi i dettagli per

ogni property). Per esporre una Thing tuya è necessario configurare il server con un campo tuyaUrl corrispondente a *"https://openapiOP.tuyaRG.com/v1.0/devices/"* dove OP e RG contengono i valori corretti come spiegato nella sezione 3. È inoltre possibile specificare un ulteriore URI, come già possibile con i server HTTP / HTTPS classici, come base per ogni action e ogni evento non gestiti da Tuya. Una volta fatto questo, il server HTTP riconoscerà l'URL e configurerà le Affordance Interaction delle properties della Thing che si vorrà esporre per avere due form, uno per la readproperty ed uno per la writeproperty. Il form per la readproperty avrà il seguente URL: *"base/THINGID/status"* dove al posto di *base* verrà inserito l'URL base fornito per il server e THINGID verrà sostituito con il nome della Thing esposta (corrispondente all'id della Thing nei server Tuya). Il form, inoltre, conterrà anche un riferimento al nome della property per dare possibilità poi al client di filtrare i dati necessari. La chiamata per la readproperty è sempre di tipo GET. Il form per la writeproperty invece ha come URL *"base/THINGID/-commands"* dove base è l'URL fornito al server e THINGID è il nome della Thing espresso nel TD e che corrisponde all'ID della Thing nei server Tuya. Anche questo form contiene il nome della property per poter far compilare correttamente la richiesta al client. Tutte i form per la writeproperty usano la POST per essere eseguite. Gli stessi URL vengono poi riproposti sia per la readAllProperties che per la writeAllProperties dato che, come già discusso, le API Tuya non fanno distinzione per lettura o scrittura di singole properties. L'unica differenza tra i form delle readAllProperties e writeAllProperties e quelli delle properties normali è il campo propertyname che, nel caso del form per leggere/scrivere tutte le properties, hanno "all" al posto del nome della property da leggere. Resta comunque possibile generare action ed eventi personalizzati che funzioneranno normalmente come per le Thing esposte in locale ed è quindi compito del servient che espone implementarle correttamente.

### 4.1.3 Sicurezza

Il sistema di sicurezza per la composizione e firma del pacchetto è alquanto complesso e, spesso, soggetto a cambiamenti. Per ogni pacchetto si richiede che nell'header vengano messi i seguenti campi:

- **t** Valore in stringa del momento in cui la chiamata viene composta (funzione Date.now()). C'è un range di validità per il time (non esplicitamente segnalato) per cui se si usa un valore t più vecchio di qualche secondo la chiamata viene annullata.

- **client_id** Il valore in stringa della chiave assegnata al momento della creazione del progetto Cloud a cui le API faranno richiesta per l'autenticazione.

- **sign_method** Questo campo deve per forza essere impostato a "HMAC-SHA256". Dato che è fisso non c'è una vera ragione per richiederlo al momento ma può essere sfruttato in futuro con possibili evoluzioni del servizio.

- **sign** Questo è il campo più complesso da compilare: il body della richiesta deve essere trasformato in stringa e, successivamente, in una Hash tramite l'algoritmo *sha256* e digest *hex*. Questa stringa verrà chiamata bodyStr da ora per semplicità. E richiesto, poi, che venga formata una stringa composta da chiave del progetto Cloud + token + istante in cui viene fatta la richiesta (identico al valore t) + metodo della chiamata HTTP + *newline* + la bodyStr + *newline* + una stringa di header formata in precedenza di cui verrà discusso nella sezione 4.1.3(questa stringa, per comodità, verrà chiamata da ora in poi extraHeaderStr) + *newline* + una stringa composta dai valori della query presente nell'URL (tutto ciò che viene dopo il '?' nell'url) in ordine alfabetico o l'URL (esclusa la base che è la stessa utilizzata per la configurazione del server) nel caso la query non fosse presente. Questa nuova stringa per semplicità da ora verrà chiamata headerStr. La stringa headerStr deve poi essere autenticata tramite HMAC con l'algoritmo SHA-256 e come chiave la chiave segreta ottenuta alla creazione del progetto Cloud di Tuya. L'autenticazione usa inoltre un digest hex. La stringa risultante da questo processo di autenticazione viene poi trasformata per avere tutti i caratteri maiuscoli ed infine può essere utilizzata come valore per il campo sign.

- **access_token** Il token necessario per la comunicazione con il server.

### Ottenimento del token

La richiesta del token deve essere fatta la prima volta (in quanto esso è necessario per ogni comunicazione con le API) e, successivamente, dopo la data di scadenza dello stesso. La data di scadenza del token viene passata assieme al token al momento del suo ottenimento. Per ottenere un nuovo token il procedimento è abbastanza simile a quello delle normali richieste: si fa una richiesta al link *https://openapi.tuyaREGION.com/v1.0/token?grant_type=1* per ottenere il token per la prima volta o a *https://openapi.tuyaREGION.com/v1.0/token/REFRESH* se si vuole ottenerne uno a partire da uno scaduto. I campi REGION corrispondono alla regione dei server da interrogare e REFRESH corrisponde ad un valore ottenuto assieme alla richiesta del token. I parametri della chiamata sono gli stessi delle chiamate normali con la differenza che, nel caso il token si stesse chiedendo per la prima volta, il campo ad esso dedicato sarebbe vuoto.
Altra differenza è il campo sign che non mette il token nella richiesta e non esegue la trasformazione della query nell'URL.

Listing 4.2: Codice per la creazione della stringa *sign*

```
private requestSign(NormalRequest:boolean, requestTime:string, body
    ↪ :any, headers: any, path:string, method:string){
    const bodyHash = crypto.createHash('sha256').update(body != null
        ↪  ? body : '').digest('hex');
    let signUrl:string = "/v1.0/token?grant_type=1";;
    const headersKeys = Object.keys(headers);
    let headerString = '';
    const useToken = NormalRequest ? this.token : '';
    const _method = method != null ? method : 'GET';
    if(NormalRequest){
        const pathQuery = queryString.parse(path.split('?')[1]);
        let query:any = {}
        query = Object.assign(query, pathQuery);
        let sortedQuery:{[k:string] : string} = {};
        Object.keys(query).sort().forEach(i => sortedQuery[i] =
            ↪ query[i]);
        const qs = queryString.stringify(sortedQuery)
        signUrl = decodeURIComponent(qs?`${path.split('?')[0]}?${qs
            ↪ }`:path);
    }
    const endStr = [this.key, useToken, requestTime, [_method,
        ↪ bodyHash, headerString, signUrl].join('\n')].join('');
    let sign = crypto
        .createHmac('sha256', this.secret)
        .update(endStr)
        .digest('hex')
        .toUpperCase();
    return sign;
}
```

**Problematiche delle API**

La creazione della richiesta Tuya presenta, al momento, due principali problematiche non dipendenti dal mio lavoro:

1. **extraHeaderStr** Questa stringa è al momento vuota. Secondo le specifiche date nella pagina di documentazione questa stringa dovrebbe contenente il valore di tutti gli altri header non necessari che si sono voluti aggiungere al pacchetto la (stringa dovrebbe essere composta da *"header1:value1 + newline + header2:value2 + newline +...*). Si dovrebbe inoltre aggiungere un nuovo header necessario al pacchetto. L'header si chiamerebbe "signHeaders" e conterrebbe una stringa formata dal nome di tutti gli header aggiuntivi separati da un segno ":". Il problema di questa aggiunta è che, se inserita nel pacchetto, non funziona e fa restituire al server sempre risposte di errore. In effetti, al momento, nemmeno la libreria ufficiale implementa la cosa ed è quindi stato necessario eliminare il passaggio in fase di stesura di codice.

2. **bodyStr** Il problema legato a questo campo è il fatto che, anche quando il body non dovrebbe essere presente (Es. chiamate GET) il server Tuya pretende comunque che il body corrisponda a '' sia nella chiamata sia nella stringa firmata degli header. Inserire body nelle chiamate di tipo GET, seppur non espressamente vietato, non è considerata una pratica da seguire e molte librerie che implementano HTTP lo impediscono. Al momento il problema è stato aggirato usando una stringa vuota al posto di un oggetto vuoto quando si fa riferimento al campo body per la creazione della bodyStr. Si noti però che questo è un *work-around* e non una soluzione ufficialmente supportata e non è quindi chiaro se questa soluzione resterà valida a seguito di possibili futuri aggiornamenti.

Le problematiche descritte sono quelle che al momento della stesura di questo documento sono presenti. Non si assicura quindi che in futuro le cose non possano cambiare necessitando di aggiornamenti al bindig o fix di varia natura.

### 4.1.4 http

In questo file sono presenti i form che vengono usati dal client per comporre la chiamata alla Thing o i form necessari alla configurazione del server. È stato necessario, in questo caso, aggiungere un form chiamato *"tuyaForm"* che estendesse l'HttpForm già presente aggiungendo un campo per la propertyName. Questo campo è particolarmente utile per le creazioni delle richieste o il filtraggio delle risposte come spiegato in sezione 4.1.1. È stato poi necessario aggiungere una tipologia di file di configurazione per i server, *TuyaConfig*, per dare la possibilità

di aggiungere l'indirizzo dei server Tuya da contattare per le richieste. Questo file estende il file HTTPConfig. Resta comunque possibile specificare tutti i campi precedentemente definibili compreso l'indirizzo personalizzato, *BaseUri*, che però si applica solo ad Action ed Events che non sono supportati dal binding.

### 4.1.5 Risposte dei server Tuya

Un altro rilevante problema delle chiamate ai server Tuya è costituito dalle risposte che esse restituiscono: al contrario delle normali richieste HTTP che segnalano gli errori con un codice 4xx o 5xx (con xx che varia a seconda dell'errore specifico), i server Tuya rispondono sempre con un codice 200. Nel caso ci fosse poi un problema di qualsiasi tipo verrebbe segnalato nella risposta ottenuta. Infatti, la risposta dei server Tuya, è sempre un oggetto contente un campo "success" che contiene un valore booleano (true se la chiamata ha avuto successo, false se la chiamata ha avuto problemi) ed un campo variabile (si ottiene infatti "result" se la chiamata ha avuto successo e "msg" se la chiamata non è stata conclusa positivamente. Il campo result contiene poi il risultato della chiamata, vuoto se la chiamata non ne avesse, mentre il campo msg contiene una stringa con un messaggio di errore in plain Text. I messaggi di errore sono piuttosto generici e spesso non descrivono adeguatamente il problema). È quindi stato necessario aggiungere alle chiamate nuovi controlli per verificarne l'avvenuta conclusione con successo. In particolare viene controllato che il valore di success sia true altrimenti viene generata un eccezione che allega il messaggio ottenuto dai server Tuya. I controlli sono stati aggiunti nelle chiamate del client e nelle chiamate per l'ottenimento del token.

## 4.2 Tecnologie

### 4.2.1 Linguaggio di programmazione

#### Typescript

Il progetto node - WoT è scritto in Typescript, linguaggio che espande Javascript con funzionalità legate soprattutto alla tipizzazione delle variabili (del tutto assente in Javascript) e alla semplificazione dell'uso del paradigma OOP per la programmazione. La semplificazione è dovuta alle più vaste possibilità di creazione delle classi con l'opzione di rendere metodi e variabili private o protette. Il Tpescript ha poi bisogno di essere compilato e ne risulta un codice Javascript. In questo caso il compilatore usato è quello ufficiale fornito da Microsoft nella sua versione 3.7.5 e nel codice sottostante sono presenti le impostazioni di compilazione usate per il progetto.

```
{
    "compilerOptions":{
        "target": "es5",
        "lib": ["es2015","dom"],
        "module": "commonjs",
        "outDir": "dist",
        "alwaysStrict": false,
        "sourceMap": false,
        "noImplicitAny": true,
        "removeComments": false,
        "noImplicitUseStrict": true,
        "noLib": false,
        "preserveConstEnums": true,
        "experimentalDecorators": true,
        "declaration": true,
    },
    "include": [
        "src/**/*"
    ]
}
```

#### Node

Il codice compilato è poi stato eseguito tramite NodeJS nella sua versione 12.21.0. NodeJS è un interprete Javascript runtime che viene eseguito sulla console e sfrutta il motore Javascript V8 di Chrome. NodeJS è particolarmente comodo per l'implementazione di server Web data la possibilità di usare sia per il front end che per il back end lo stesso linguaggio di programmazione. In aggiunta usare Javascript, linguaggio interpretato, aiuta a rendere il codice più facilmente trasportabile su diverse macchine. Anche per questo è stato scelta questa piattaforma per l'implementazione dello standard WoT: in questo modo è facile portare lo stesso software su una moltitudine di diverse architetture.

### 4.2.2 Librerie

Le librerie utilizzate dal Progetto node - WoT sono diverse e si eviterà, in questo documento, di descrivere quelle che non vengono direttamente usate dal progetto.

- **node-fetch** Una semplice libreria per la fetch delle API. È stata utilizzata per creare richieste soprattutto riguardanti l'ottenimento del token dai server Tuya.
- **JSON** Libreria di default per la gestione dei file JSON. Utile per analizzare i dati ricevuti dalle API Tuya e creare i pacchetti conformi alle richieste delle API Tuya.

È stato poi necessario aggiungere altre librerie per riuscire ad implementare il binding in maniera soddisfacente:

- **crypto** Una libreria che implementa funzionalità di crittografia. È stata usata per firmare e criptare le informazioni nel formato richiesto dalle API Tuya.
- **query-string** Una libreria che semplifica lo spezzamento di URL nelle sue componentistiche fondamentali creando un oggetto con all'interno tutti i dati inseriti nella query dell'URL.

# Capitolo 5

# Validazione

Verrà ora illustrato un prototipo di interazione di una Thing Tuya esposta come Thing WoT tramite un servient Node-WoT ed una Thing WoT classica. Le due Thing comunicheranno e si scambieranno dati tra loro per creare un ambiente di "Smart Home" integrato. Verrà prima presentato un semplice esempio di intercomunicazione tra le Thing, successivamente si discuterà l'implementazione della stessa ed infine verrà discusso l'overhead dovuto all'esposizione della Thing Tuya come Thing WoT.

## 5.1 Integrazione per una Smart Home

L'obiettivo di questa validazione è di mostrare come una Thing Tuya possa, tramite il binding, integrarsi in un ambiente WoT. Per fare questo e stata implementata una semplice interazione d'esempio che sfruttasse una Thing WoT creata ad hoc ed uno *Smart Bulb RGB* Tuya. Le due Thing interagiranno tra loro per fare in modo che il colore della lampadina sia diverso a seconda della temperatura della stanza (rosso se caldo, blu se freddo e bianco altrimenti). L'interazione è a solo scopo dimostrativo delle possibilità di integrazione dei diversi dispositivi Tuya sul mercato in un ambiente WoT.

### 5.1.1 Modello di interazione

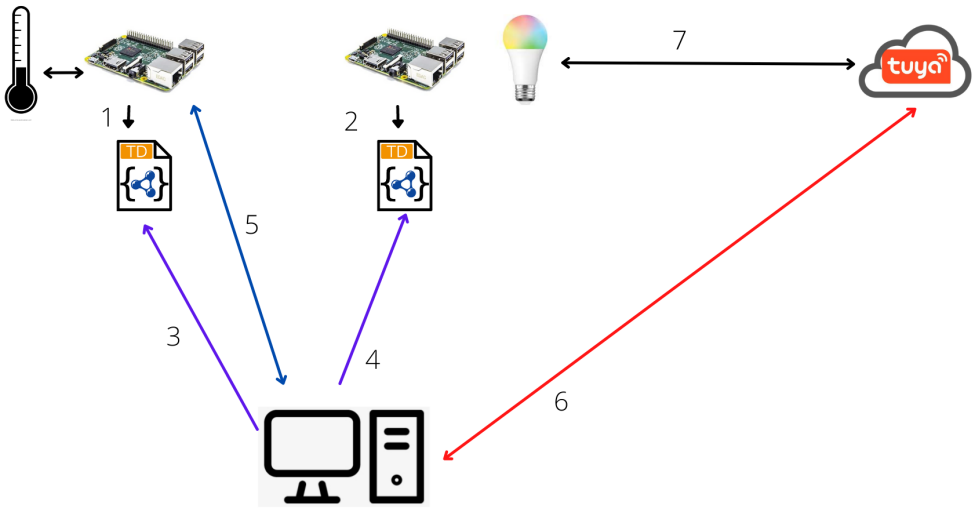


Figura 5.1: Schema di interazione dell'esempio di validazione

Nella figura 5.1 viene illustrato lo schema di interazione tra le Thing dell'esempio di validazione

1. Il primo raspberry espone la Thing per il sensore di temperatura come una normale Thing WoT con una sola property per leggere la temperatura rilevata.

2. Il secondo raspberry espone la Thing Tuya. La TD ha mappato, in questo caso, tutte le property della Thing e permette quindi di utilizzarla al pari dell'applicazione ufficiale.

3. L'applicazione di mashup consuma la Thing sensore tramite un client HTTP. La TD è reperibile ad un indirizzo di rete locale.

4. L'applicazione di mashup consuma la Thing Tuya tramite un client HTTP. La TD è reperibile ad un indirizzo di rete locale diverso da quello della Thing sensore. Il client HTTP non deve necessariamente essere configurato per poter comunicare con i server Tuya.

5. L'applicazione di mashup comunica con il raspberry che espone la Thing WoT per sottoscrivere gli eventi di temperatura emessi dalla Thing sensore.

In questo modo l'applicazione di mashup può essere notificata quando c'è un cambio di temperatura drastico.

6. Allo scoccare degli eventi l'applicazione di mashup comunica con i server Tuya (tramite il binding Tuya del node - WoT) per modificare lo stato della Thing Tuya. In questo caso il client HTTPS che viene usato per la comunicazione deve essere configurato per rispettare le richieste di creazione del pacchetto dei server Tuya.

7. I server Tuya comunicano con la Thing Tuya per cambiarne lo stato e rispondere all'applicazione di mashup. Il client HTTPS configurato Tuya controllerà la risposta e genererà un'eccezione nel caso fosse negativa.

## 5.2 Implementazione

L'interazione è stata sviluppata creando due servient, uno per ogni Thing esposta: il primo gestisce anche la Thing che espone mentre il secondo serve solo per esporre la TD della Thing Tuya. Esiste poi un terzo servient che agisce da mashup application e, sottoscrivendosi agli eventi del sensore, cambia il colore della lampadina Tuya.

### 5.2.1 Thing WoT

Per la Thing WoT è stato implementato un semplice sensore di temperatura controllato da un RaspBerry Pi zero 2[18] configurato per avere IP statico nella rete locale. Il sensore utilizzato invece è un Adafruit PCT2075[1]. La lettura della temperatura viene fatta tramite la libreria Python ufficiale fornita da Adafruit. Uno script python fa il polling della temperatura ogni secondo e scrive il dato su un file dedicato. Successivamente una funzione javascript legge il dato dal file per esporlo. La TD del sensore esposto è molto semplice e presenta una sola Property: *"Temperature"* che rappresenta la temperatura letta dal sensore in valore intero e varia tra i -100 ed i 100 gradi. Il servient è stato configurato per esporre la Thing usando il protocollo HTTP ed il server è configurato per restare in ascolto sulla porta 5000. L'apertura delle porte per rendere la Thing consumabile anche da dispositivi terzi è poi stata fatta usando il software nginx[25]

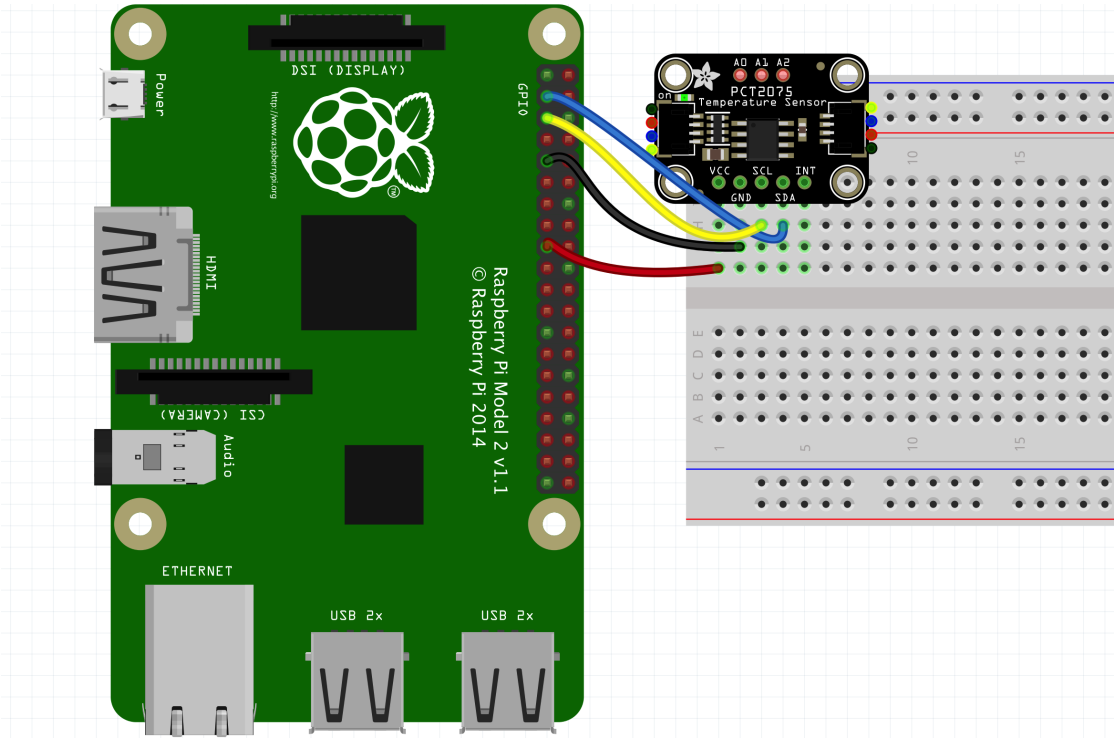


Figura 5.2: Collegamento del sensore al Raspberry

Non è disponibile nessuna azione ma la Thing genera tre eventi:

- **HighTemperature** per quando la temperatura è sopra i 30 gradi.
- **NormalTemperature** per quando la temperatura è tra i 30 e gli 0 gradi.
- **LowTemperature** per quando la temperatura è sotto gli 0 gradi.

### 5.2.2 Thing Tuya

La Thing Tuya espone tutte le properties che la Thing mette a disposizione. La Thing non permette azioni o eventi in quanto non previsti dal produttore. Quest'ultimo impone che ogni interazione sia fatta tramite il cambiamento dello stato delle varie proprietà. Il servient è hostato su un raspberry pi configurato per avere IP statico nella rete locale. La lampadina utilizzata è una generica lampadina rebrandizzata comprata online[42]. Il servient è stato configurato per esporre la Thing usando il protocollo HTTP ed il server è configurato per restare in ascolto sulla porta 5000. Dato però che la comunicazione con i server Tuya è sempre

compito del client è necessario, per consumarla, aver configurato anche una *ClientFactory* HTTPS. L'apertura delle porte per rendere la Thing consumabile anche da dispositivi terzi è infine stata resa possibile usando il software nginx[25]

### 5.2.3 Servient applicazione di mashup

Il terzo servient simula un'applicazione di mashup che combina le due Thing esposte. È configurato con due *ClientFactory*: HTTP ed HTTPS per comunicare con entrambe le Thing. Esegue un semplice script (che può comunque essere espanso a piacimento fino a diventare un'applicazione completa) che consuma le due Thing localmente esposte e, sottoscrivendosi agli eventi del sensore, cambia il colore della lampadina a seconda della situazione in cui ci si trova.

## 5.3 Analisi delle prestazioni

Si è misurato l'overhead che il binding Tuya comporta nell'utilizzo della Thing esposta per verificare che tale delay aggiuntivo non fosse esageratamente elevato da limitarne i casi d'uso. Per raccogliere i dati è stato deciso di svolgere 4 tipi di operazioni:

- Lettura dati "leggera": la lettura di un booleano dai server Tuya.
- Lettura dati "pesante": la lettura di un oggetto json dai server Tuya.
- Scrittura dati "leggera": la scrittura di un booleano sui server Tuya.
- Scrittura dati "pesante": la scrittura di un oggetto json sui server Tuya.

Ogni tipo di operazione è stata ripetuta 1000 volte sia tramite l'uso del progetto node - WoT (e quindi passando dal binding) sia tramite la libreria ufficiale Tuya chiamata tuya-connector-nodejs. Nei dati raccolti sulle chiamate tramite node - Wot è stato poi diviso il tempo dovuto alla chiamata effettiva (dal momento in cui viene effettivamente contattato l'enpoint a quando i dati sono stati ricevuti eliminando dunque tutta la parte di creazione della chiamata) e gli overhead dovuti alle varie componenti del progetto: Core e Client. Nelle chiamate invece fatte tramite tuya-connector-nodejs è stata fatta distinzione tra la chiamata effettiva e l'overhead dovuto alla preparazione della stessa. In entrambi i casi l'ottenimento del token è stata segnato come overhead anche se necessitava di una chiamata aggiuntiva. I dati sono stati raccolti usando la funzione "process.hrtime()" di node che restituisce un array [secondi, nanosecondi] rappresentante il tempo trascorso da un arbitrario momento nel passato [26]. Per testare la Thing tramite node - WoT è stato poi necessario prima esporla. L'esposizione è sempre stata svolta in

locale dalla stessa macchina che ha poi effettuato le richieste. Sono stati, infine, raccolti i tempi impiegati poi dalla partenza della chiamata effettiva all'uscita del pacchetto dall'interfaccia di rete. La cattura di questi dati è stata fatta usando Wireshark. I dati non mostrano differenze significative tra node - WoT e la libreria ufficiale ed hanno sempre misurato circa un centesimo di secondo di tempo.

### 5.3.1 lettura dati leggera

La lettura dati delle API Tuya non distingue le singole informazioni ma restituisce un unico array di oggetti che contiene tutti le proprietà della Thing. Per questa ragione i dati raccolti tramite node - WoT contengono un overhead aggiuntivo dovuto al fatto che il client deve analizzare l'array per restituire solo l'informazione richiesta. La stessa procedura che invece deve essere fatta successivamente se si sfrutta la libreria ufficiale. Sempre per la stessa ragione la lettura di dati pesanti tramite la libreria ufficiale riporta gli stessi dati indicati in questa sezione.

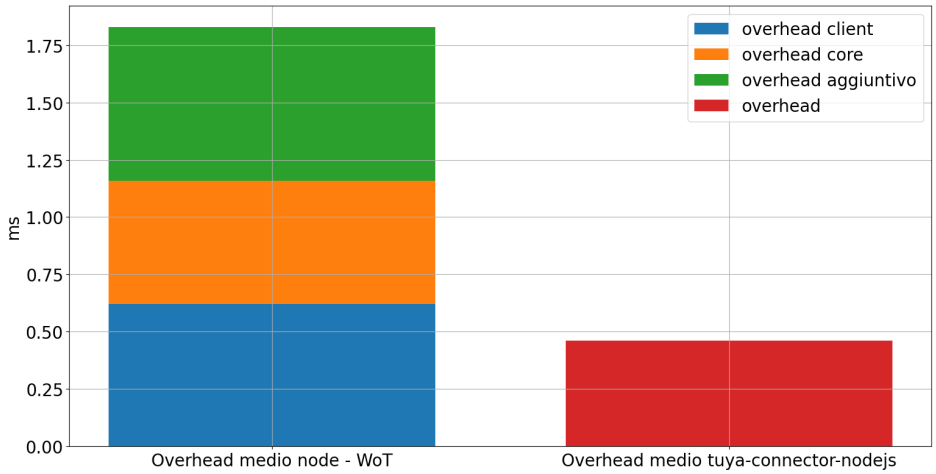


Figura 5.3: L'overhead di 1000 letture di dati "leggeri"

Nella figura 5.3 possono essere visti i dati delle medie delle chiamate node - WoT e Tuya. Essi non mostrano una confidenza statistica in quanto la presenza della chiamata 0 ne distorceva la rilevanza oltre che la leggibilità del grafico.
Analizzando invece le chiamate più significative delle 1000 svolte si più notare che quella con overhead più elevato è la numero 0 (130.15 ms) che deve fare richiesta per il token che verrà poi utilizzato da tutte le chiamate successive. Le chiamate hanno un overhead medio di 1.82 ms con il componente client che ha un overhead

medio leggermente più alto 0.62 ms contro il core che occupa 0.53 ms (il recupero del token dai server o dalla memoria viene sempre contato come overhead ma non attribuito a nessun componente). Le restanti chiamate ritenute significative sono le seguenti:

- 810: la chiamata che ha impiegato più tempo in totale: 5103.70 ms. Il lungo tempo di attesa è dovuto principalmente alla lenta risposta dei server Tuya: infatti la chiamata effettiva è durata 5102.02 con circa 1.7 ms di overhead.
- 916: la chiamata che ha impiegato meno in assoluto: 79.42 ms, con 78 ms di chiamata effettiva ed un overhead di 1.4 ms circa.
- 406: la chiamata con il minor overhead. Circa 1.3 ms di delay rispetto alla chiamata effettiva che ha poi impiegato 97.33 ms.

Lo stesso tipo di chiamata eseguito tramite tuya-connector-nodejs ha fornito i seguenti risultati:

- La chiamata che ha richiesto più tempo in assoluto ed anche con il tempo effettivo più elevato è stata la numero 230 con 5103.26 ms di tempo totale e 5102.92 di tempo effettivo, 1.7 ms di overhead.
- 370: la chiamata che ha impiegato sia meno tempo in assoluto che di chiamata effettiva: 78.98 ms di chiamata totale e 78.66 ms di chiamata effettiva con meno di un ms di overhead.
- La chiamata che più delle altre ha avuto overhead è, anche in questo caso, la chiamata 0 con 104.79 ms di chiamata effettiva ma 216.88 ms di chiamata totale. Anche in questo caso è significativo che sia la chiamata 0: essa deve recuperare il token e occupa 112.09 ms di overhead in totale.
- La chiamata con meno delay in assoluto è la numero 502, con appena 0.13 ms di overhead e 119.88 ms di chiamata complessiva.

Di media l'overhead totale è stato di 0.46 ms: decisamente più basso di quello ottenuto con node - WoT. È comunque importante ribadire che parte di questo overhead aggiuntivo è dovuto al fatto che node - Wot si occupa anche di filtrare il dato specifico richiesto.

### 5.3.2 Lettura dati pesante

Come già discusso le API Tuya restituiscono tutte le informazioni relative alla Thing ed è quindi inutile discutere l'overhead relativo a tuya-connector-nodejs.
La situazione cambia se si considera il node - WoT: in questo caso il dato deve

essere cercato all'interno dell'array. Il test è quindi stato ripetuto ma, anche in questo caso, non sono state riscontrate differenze significative: la media di overhead si attesta su 1.71 ms (addirittura leggermente più bassa rispetto alla precedente) con un picco di 121.16 ms dovuto all'ottenimento del token con una minima invece di 0.67 ms.

### 5.3.3 Scrittura dati leggera

Come per la lettura, anche in questo caso, è importante notare la differenza con la libreria ufficiale che pretende in input un oggetto formattato secondo lo standard descritto dalle API. Al contrario, nel progetto node - WoT è il client a creare l'oggetto formattato nella maniera corretta. Questo non può che portare ad un appesantimento del processo che però dovrebbe essere presente, seppur in parte, anche in un eventuale progetto che usasse tuya-connector-nodejs.
Per quanto riguarda il progetto node - WoT la media di overhead si attesta sui 1.59 ms, 0.17 dei quali dovuti al client e 0.43 al core. Altri dati significativi invece sono:

- La chiamata 592 che è quella con i tempi di esecuzione maggiori: 5133.24 ms ma pressochè totalmente dovuti alla lenta risposta dei server Tuya 5132.01 ms di chiamata.
- La chiamata 796 che ha i tempi di esecuzione minori: 86.55 ms ma che ha comunque un overhead in linea con quello medio: 1.4 ms circa.
- La chiamata 0 che, come nei casi precedenti è quella con maggior overhead dovuto alla richiesta del token che ha impiegato in tutto 252.56 ms di cui 107.14 di overhead e di questi ultimi solo poco più di 5 dovuti ai componenti software del progetto node - Wot.
- La chiamata 860 che invece ha avuto il minor overhead: circa 0.5 ms.

Usando invece tuya-connector-nodejs l'overhead è di 0.52 ms di media. Altri dati significativi sono:

- La chiamata 439 che ha avuto la durata massima: 5081.742812998593 ms con però appena 0.36 ms circa di overhead.
- La chiamata 925 che ha avuto la durata minima: 57.25 ms con 0.3 ms di overhead circa.
- La chiamata 0 che ha avuto il maggior overhead: 151.53, sempre dovuto all'ottenimento del token.
- La chiamata 566 che ha avuto il minor overhead: 0.13 ms.

### 5.3.4 Scrittura dati pesante

I dati ottenuti dalla scrittura pesante sono in linea con quelli riscontrati tramite le precedenti misurazioni: la media del progetto node - WoT si attesta a circa 1 ms in più rispetto a quella di tuya-connector-nodejs (1.39 contro 0.47) e, come nei precedenti casi, è da imputare, almeno in parte, alla formattazione svolta da node - WoT invece assente in tuya-connector-nodejs. L'immagine 5.4 rappresenta l'overhead ottenuto con 999 scritture di dati pesanti sui server Tuya tramite la libreria ufficiale Tuya e node - WoT. È stato escluso il primo valore di ottenimento del token . Dalla parte di node - WoT sono stati poi tagliati sei valori "anomali". Quattro di questi erano compresi tra 6 e 7 ms, mentre 2 di questi tra 11 e 12 ms. Queste misurazioni sono state escluse per permettere una più facile lettura del grafico.

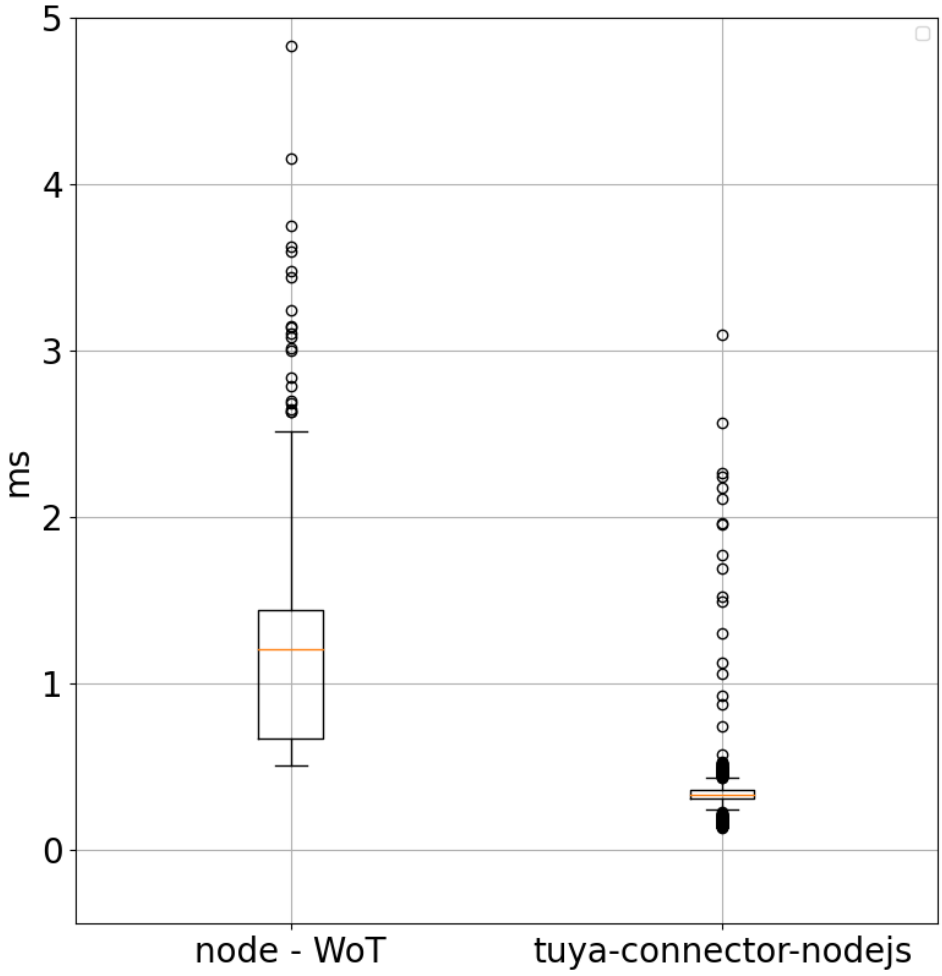


Figura 5.4: Overhead di 999 scritture ”pesanti” effettuate con tuya-connector-nodejs e node - WoT

# Capitolo 6

# Conclusioni

In questa tesi si è mostrato un possibile modello di integrazione delle Thing Tuya con lo standard WoT esposto dal W3C. L'integrazione è a livello client. È quindi-quest'ultimo ad agire per convertire i dati internamente prima di spedirli all'utente o ai server Tuya. L'exposer ha solo il compito di esporre una TD correttamente compilata. L'obiettivo di questo progetto è di espandere il catalogo di Thing compatibili con lo standard WoT per fare in modo di diffondere ulteriormente lo standard ed ampliare l'ecosistema e l'interoperabilità delle Thing già presenti.

## 6.1 Stato attuale del progetto

Al momento il binding supporta tutte le Thing Tuya e non comporta un overhead di comunicazione che ne impedisca l'utilizzo. Il progetto però deve tenere conto di alcune limitazioni non dipendenti dalla mia implementazione.

### 6.1.1 Limiti

#### A Tuya piace cambiare

Purtroppo le API Tuya sono in continuo mutamento per nessuna ragione apparente. Da quando è stato iniziato lo sviluppo sono state cambiate 3 volte e questo rende lo stato del progetto alquanto instabile.
Ad ogni grosso cambio invalidante delle vecchie metodologie di comunicazione, Tuya mantiene attivi anche i vecchi metodi per i progetti Cloud creati precedentemente. Questo implica che un'eventuale manutenzione dovrebbe richiedere anche il riconoscimento della versione delle API usate e quindi si avrebbe un aggiuntivo overhead dovuto alla necessità di questo controllo.

**Inconsistenza della documentazione**

Come spiegato nella sezione 4.1.3 diverse incongruenze ed errori sono presenti nelle API. Eventuali cambi di controlli o fix di bug ora presenti potrebbero dover richiedere una manutenzione del codice indipendente dal cambio di versione delle API. È inoltre importante ricordare che, per ora, è possibile non inviare il body nelle chiamate "GET" solo perchè è stato sfruttato un errore nei server Tuya. La libreria ufficiale infatti prevede che il body venga inviato sempre e comunque. Se in futuro il bug che permette l'invio di richieste senza body dovesse essere risolto l'unico fix possibile dovrebbe prevedere di andare contro lo standard HTTP.

## 6.2 Sviluppi futuri

Sarà sicuramente necessaria una costante manutenzione del progetto. Inoltre, questa tesi si è concentrata solo sulla parte di interazione diretta con la Thing ignorando tutta la parte di aggiunta e configurazione di quest'ultima. I server Tuya mettono a disposizione delle API per i progetti cloud anche per consentire la configurazione ma prevedono che esse siano legate ad un account. Il node - WoT al momento non prevede la gestione di account ed è quindi doveroso imbastire una discussione sull'eventualità di gestire questo aspetto e le metodologie per farlo. Se non collegate ad un account, le Thing Tuya non possono funzionare. Lo stesso discorso vale per gli aggiornamenti che le Thing supportano: essi infatti vengono a loro volta distribuiti tramite API che possono essere integrate con futuri sviluppi. Il lavoro si è poi fermato al generico funzionamento delle Thing ma sono messe a disposizione (spesso con servizi a pagamento) una moltitudine di altre API. Queste variano dalla gestione delle notifiche alla collezione automatica di dati statistici per lo sfruttamento in diversi contesti (analisi del traffico o dei consumi industriali e così via). È possibile aggiungere la compatibilità con queste ultime senza troppo lavoro dato che, alla base, funzionano come quelle già integrate.

### 6.2.1 Migliorie future del node - WoT

Il node - WoT ha previsto un URI personalizzato che permetta la lettura e la scrittura di tutte le properties. Ad oggi il progetto non sfrutta però tale URI per eseguire le operazioni ad esso dedicate ma, al contrario, effettua un ciclo che, per ogni property, esegue una chiamata. Questo aggiunge un overhead non indifferente al binding Tuya. Tuttavia è stato già predisposto l'indirizzo per permettere le operazioni di lettura di tutte le properties e, quando esso verrà utilizzato da node - WoT, sarà sufficiente una sola chiamata ai server Tuya. L'overhead verrà così fortemente limitato e riportato nella media di quanto già descritto nel capitolo 5.

# Bibliografia


[1] Nxp Is A. Overview — adafruit pct2075 temperature sensor — adafruit learning system, 2021.

[2] Kevin Ashton et al. That 'internet of things' thing. *RFID journal*, 22(7):97–114, 2009.

[3] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[4] Timothy Chou. *Precision-Principles, Practices and Solutions for the Internet of Things*. McGraw-Hill Education, 2017.

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

[6] eclipse. Node-wot github page, 2021.

[7] Mahmoud Elkhodr, Seyed Shahrestani, and Hon Cheung. The internet of things: new interoperability, management and security challenges. *arXiv preprint arXiv:1604.04824*, 2016.

[8] Roy Thomas Fielding. Architectural styles and the design of network-based software architectures, 2000.

[9] Dominique Guinard. What is the web of things? – web of things, 2021.

[10] Dominique Guinard, Iulia Ion, and Simon Mayer. In search of an internet of things service architecture: Rest or ws-*? a developers' perspective. In *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 326–337. Springer, 2011.

[11] Dominique Guinard and Vlad Trifa. Towards the web of things: Web mashups for embedded devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain*, volume 15, page 8, 2009.

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

[13] jasonacox. tinytuya, 2021.

[14] Andreas Kamilaris, Semih Yumusak, and Muhammad Intizar Ali. Wots2e: A search engine for a semantic web of things. In *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pages 436–441. IEEE, 2016.

[15] Foughali Karim, Fathalah Karim, et al. Monitoring system using web of things in precision agriculture. *Procedia Computer Science*, 110:402–409, 2017.

[16] Tim Kindberg, John Barton, Jeff Morgan, Gene Becker, Debbie Caswell, Philippe Debaty, Gita Gopal, Marcos Frid, Venky Krishnan, Howard Morris, et al. People, places, things: Web presence for the real world. *Mobile Networks and Applications*, 7(5):365–376, 2002.

[17] Sameer Kumar, Eric Swanson, and Thuy Tran. Rfid in the healthcare supply chain: Usage and application. *International journal of health care quality assurance*, 22:67–81, 02 2009.

[18] Raspberry Pi Ltd. Raspberry pi zero 2 w – raspberry pi, 2021.

[19] Rwan Mahmoud, Tasneem Yousuf, Fadi Aloul, and Imran Zualkernan. Internet of things (iot) security: Current status, challenges and prospective measures. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 336–341. IEEE, 2015.

[20] Luca Mainetti, Vincenzo Mighali, and Luigi Patrono. A software architecture enabling the web of things. *IEEE Internet of Things Journal*, 2(6):445–454, 2015.

[21] Makkox. Curiosity rover (@marscuriosity) / twitter, 2021.

[22] Catherine C Marshall and Frank M Shipman. Which semantic web? In *Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, pages 57–66, 2003.

[23] Sujith Samuel Mathew, Yacine Atif, Quan Z Sheng, and Zakaria Maamar. Web of things: Description, discovery and integration. In *2011 International conference on Internet of Things and 4th International Conference on Cyber, Physical and Social Computing*, pages 9–15. IEEE, 2011.

[24] Ismail Nadim, Yassine Elghayam, and Abdelalim Sadiq. Semantic discovery architecture for dynamic environments of web of things. In *2018 International Conference on Advanced Communication Technologies and Networking (CommNet)*, pages 1–6. IEEE, 2018.

[25] NGINX. Nginx — high performance load balancer, web server, reverse proxy, 2021.

[26] Node. Node process documentation page, 2021.

[27] KA Patil and NR Kale. A model for smart agriculture using iot. In *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, pages 543–545. IEEE, 2016.

[28] S Mohamed Rabeek, Han Beibei, and Kevin TC Chai. Design of wireless iot sensor node & platform for water pipeline leak detection. In *2019 IEEE Asia-Pacific Microwave Conference (APMC)*, pages 1328–1330. IEEE, 2019.

[29] Mohammad Abdur Razzaque and Siobhan Clarke. Smart management of next generation bike sharing systems using internet of things. In *2015 IEEE First International Smart Cities Conference (ISC2)*, pages 1–8. IEEE, 2015.

[30] Michele Ruta, Floriano Scioscia, and Eugenio Di Sciascio. Enabling the semantic web of things: framework and architecture. In *2012 IEEE Sixth International Conference on Semantic Computing*, pages 345–347. IEEE, 2012.

[31] Martin Serrano, Payam Barnaghi, Francois Carrez, Philippe Cousin, Ovidiu Vermesan, and Peter Friess. Internet of things iot semantic interoperability: Research challenges, best practices, recommendations and next steps. *IERC: European Research Cluster on the Internet of Things, Tech. Rep*, 2015.

[32] Lu Tan and Neng Wang. Future internet: The internet of things. In *2010 3rd international conference on advanced computer theory and engineering (ICACTE)*, volume 5, pages V5–376. IEEE, 2010.

[33] Vlad Trifa, Samuel Wieland, Dominique Guinard, and Thomas Michael Bohnert. Design and implementation of a gateway for web-based interaction and management of embedded devices. *Submitted to DCOSS*, pages 1–14, 2009.

[34] Tuya. Tuya website, 2021.

[35] Hans Van Der Veer and Anthony Wiles. Achieving technical interoperability. *European telecommunications standards institute*, 2008.

[36] W3C. 26 in draft-ietf-httpbis – hypertext transfer protocol wiki, 2021.

[37] W3C. Td documentation page, 2021.

[38] W3C. Wot w3c documentation, 2021.

[39] W3C. Wot w3c page, 2021.

[40] Erik Wilde. Putting things to rest, 2007.

[41] Li Da Xu, Wu He, and Shancang Li. Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*, 10(4):2233–2243, 2014.

[42] YINSAN. Yinsan lampadina intelligente wifi smart dimmerabile e27 9w 850lm multicolore lampadina, 2021.

[43] Min Zhang, Tao Yu, and Guo Fang Zhai. Smart transport system based on "the internet of things". In *Measuring Technology and Mechatronics Automation*, volume 48 of *Applied Mechanics and Materials*, pages 1073–1076. Trans Tech Publications Ltd, 3 2011.