Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Micro-WoTServient: progettazione ed implementazione di uno stack software Web of Things per microcontrollori

Relatore:
Chiar.mo Prof.
Marco Di Felice

Presentata da:
Federico Rachelli

Correlatore:
Dott.
Luca Sciullo

Sessione II
Anno Accademico 2020 / 2021

Abstract

In questa tesi triennale si discuterá la progettazione e l'implementazione di uno stack WoT Servient per dispositivi embedded per l'utilizzo nel campo dell'Internet of Things. Nella realizzazione di tale progetto si sono seguiti gli standard definiti dal consorzio W3C in materia di Web of Things. L'interfaccia del progetto si compone di una GUI con la quale l'utente puó realizzare la Thing Description associando per ogni Interaction Affordance un set di Binding Templates realizzati, tra cui HTTP, WebSocket e CoAP. Di seguito sará possibile compilare lo *sketch* ed effettuare l'upload su una board prototipale compatibile con lo stack software Arduino. La piattaforma é stata sviluppata per realizzare codice per la board Espressif ESP32, ma portabile su altri devices con caratteristiche compatibili apportando le opportune modifiche al template del microcontrollore.

Indice

Elenco delle figure

Capitolo 1

Introduzione

L'Internet of Things é un ambito nato all'inizio degli anni 2000 che estende la rete Internet ad oggetti (cose, "Things"), rendendole raggiungibili, interagibili e capaci di aggregare e combinare dati. Con l'avanzare della tecnologia e della disponibilitá di dispositivi adatti a tale scopo, il panorama delle soluzioni é cambiato rispetto alla sua prima formulazione (in cui si discuteva solamente dispositivi di tipo RFID[1]) ed é tuttora in continua evoluzione.
Nonostante molti dei limiti esistenti venti anni fa siano stati superati grazie ad una maggiore connettivitá e dalla miniaturizzazione dei microprocessori, ora ci si trova invece con il problema della frammentazione delle tecnologie utilizzate e dall'interoperabilitá tra vari sistemi IoT.
Il lavoro di standardizzazione del protocollo di comunicazione di dispositivi embedded orientati all'IoT risulta essere molto complesso, in quanto come requisito si richiede un certo livello di flessibilitá ed occorre che tutti i manifacturer e produttori software/firmware aderiscano a tale standard. Difatti, la grande problematica che si riscontra é quella che ciascun dispositivo, per la sua natura applicativa, comunica e si interfaccia con protocolli e tecnologie che piú si adattano alle proprietá dell'oggetto da connettere.
Ad esempio, un semplice attuatore funzionante a batteria richiede uno scarso dispendio energetico; esistono protocolli di comunicazione che garantiscono tale requisito, ma potrebbero non essere compatibili con altri devices che

interagiscono con altri dispositivi in rete.
Il Web of Things (abbreviato in WoT) é lo standard che definisce una serie di protocolli e modalitá di rappresentazione dei dati di dispositivi (fisici oppure virtuali) come "lingua franca" per l'intercomunicazione non solo con utenti, ma anche con altri dispositivi e/o stack software. Il W3C (organizzazione non governativa che realizza e promuove gli standard da utilizzare nel web) dal 2015 dispone di un team di lavoro che ha come obiettivo quello di ridurre la grande frammentazione che si trova ora sul mercato nel mondo dell'Internet of Things. L'obiettivo fondamentale che si pone il WoT é quello di fornire un certo livello di standardizzazione per permettere una maggiore integrazione tra protocolli di comunicazione esistenti nel web, senza sostituirli ma anzi promuovendone la diversitá, ponendosi come elemento complementare per il dialogo tra i vari casi d'uso esistenti.
Lo scopo di questa tesi é la progettazione dello stack software necessario per implementare un WoT Servient in board embedded per uso IoT, come suggerito dalle linee guida fornite dal W3C.

Questa tesi é suddivisa in due parti: la prima (capitoli 2.1 e 2.2) illustra lo stato dell'arte dell'Internet of Things, del Web of Things e delle specifiche introdotte dal W3C. Nella seconda parte si entra piú nel dettaglio del progetto WoT Servient sviluppato, mostrando i requisiti dell'architettura, le componenti realizzate e le scelte progettuali intraprese (capitolo 3), l'implementazione progettuale (capitolo 4) ed una presentazione di un caso d'uso realizzabile con tale prodotto (capitolo 5).

Capitolo 2

Stato dell'arte

2.1 Internet of Things

Con il termine Internet of Things (IoT) si definisce un insieme di dispositivi (integrati in oggetti fisici di uso quotidiano), costruiti ed ideati con lo scopo di poter intercomunicare tra di loro attraverso la rete Internet. Questa terminologia, coniata da Kevin Ashton al MIT di Boston[1] e presentata presso P&G nel 1999 , é uno dei principali oggetti di studio da parte di informatici, makers e designer. La ragione é presto detta: con l'avanzare della tecnologia e della miniaturizzazione di dispositivi informatici anche per uso prototipale, l'idea della microcomputazione é diventata sempre piú realistica e concretizzabile, dando ampio spazio alla sperimentazione.
L'idea che sta alla base é che le *persone* si spostano ed interagiscono con *oggetti* in determinati *luoghi*[2]; in un mondo sempre piú orientato all'interconnessione, si parla quindi di *presenza nel web* di tali oggetti, definiti generalmente **Things**.
Il concetto di Thing come ora noi lo intendiamo é stato plasmato negli anni con lunghe fasi di sviluppo e di ricerca. Ora con tale termine si possono riconoscere da dispositivi passivi muniti di un Tag ID (come RFID, QR code, codici a barre), dispositivi attivi per sistemi di domotica (Smart Plug, Smart Bulb, termostati, ecc..) fino ad automobili, stanze ed edifici. Il potere

del web é quello di fornire (per design) accesso ampliamente diffuso, essendo portabile anche su dispositivi mobile data dalla qualitá sempre crescente delle connessioni. Se fino a qualche anno fa Internet faceva parte della nostra vita domestica e lavorativa, ora le interazioni avvengono anche in mobilitá, nei negozi, in ristoranti e persino in parchi e strade[3]. Ora si possono trovare servizi web (come server HTTP embedded) anche in dispositivi che le persone durante i movimenti quotidiani possono incontrare, come stampanti, Smart TV o proiettori, con l'unico requisito che tali devices dispongano di una interfaccia di rete.

2.1.1 Casi d'uso e caratteristiche dei dispositivi IoT

Con il tempo il paradigma IoT ha assunto numerose interpretazioni, ampliando quindi la platea dei casi d'uso e delle applicazioni pratiche.

In commercio ora si possono trovare oggetti di uso casalingo, come elettrodomestici, attuatori, lampadine e *smart hubs* che integrano (oppure essi stessi sono) dispositivi Internet of Things. Con il tempo il campo si é dimensionato sino a soluzioni anche in campo industriale (e.g. intercomunicazioni tra macchine in una catena di montaggio), medico, edile, agroalimentare, ambientale ed in svariate altre applicazioni.

I dispositivi IoT hanno la caratteristica di essere di fatto microcalcolatori con una o piú delle seguenti caratteristiche:

- Elaborazione dati provenienti dalla sensoristica collegata
- Attuazioni di altri dispositivi come relay, dispositivi luminosi, ecc...
- Elaborazione logica di dati
- Interfacce di rete

Oltretutto, a seconda del loro campo di utilizzo le varie Things in commercio possono richiedere requisiti come il costo, dimensione e dispendio di energia elettrica contenuti. Quelle appena citate non sono caratteristiche

sempre presenti, ma dipendono fortemente dal fine ultimo del dispositivo. Ad esempio, se il device deve essere alimentabile a batteria sicuramente si richiede che consumi poca energia per massimizzare l'autonomia.

2.1.2 Architettura dei dispositivi IoT

Negli anni si sono definite molte opinioni su quanti e quali livelli utilizzare per la suddivisione dell'Internet of Things. Molti ricercatori si sono accordati[4] sull'utilizzo di tre layers (dove il livello di sicurezza é trasversale tra i definiti). In figura 2.1 si ha una rappresentazione visuale dei livelli.

- **Livello percettivo**
  Lo scopo di tale livello é quello di acquisire dati dall'ambiente esterno alla Thing con sensori oppure attuatori. Questo layer ottiene, memorizza e processa le informazioni per poi passarle al livello di rete per essere trasmesse.

- **Livello di rete**
  Il ruolo di questo layer é quello di trasmissione dei dati a dispositivi connessi ad Internet. In questa parte ci si interfaccia con i vari dispositivi della rete come Gateways, Switch e Router a seconda della tecnologia in uso (che puó essere WiFi, ZigBee, BLE, ecc...).

- **Livello applicativo**
  Il livello applicativo garantisce l'autenticazione, l'integritá, l'accesso e la confidenzialitá del dato trasmesso.

2.1.3 Eterogeneità dei devices IoT

Per definizione, l'interconnessione tra i vari devices IoT é requisito fondamentale per lo scambio di dati e l'automazione del sistema di Things. Il problema principale di tutta la tecnologia Internet of Things, ad oggi, é l'enorme frammentazione a cui si va incontro quando si decide di realizzare un

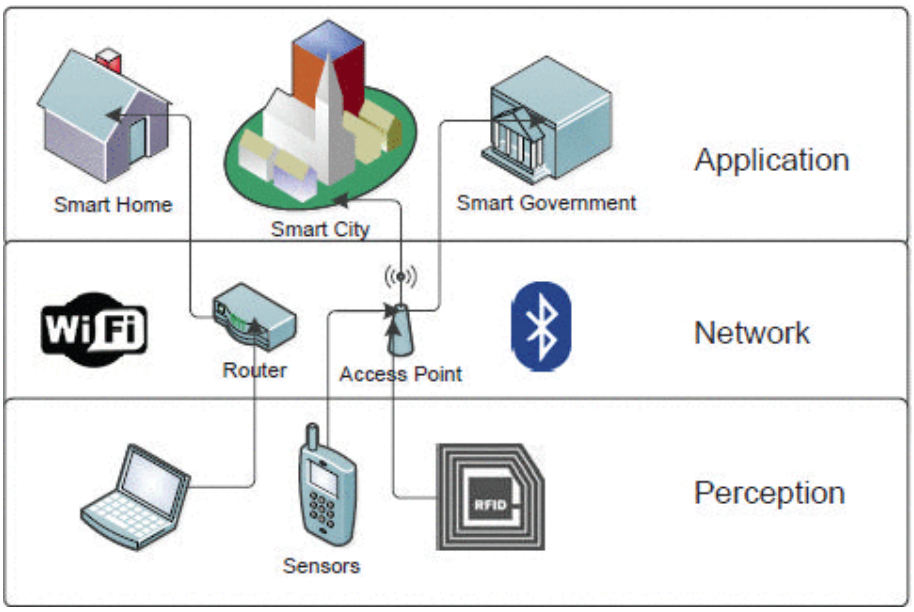

Figura 2.1: L'architettura dell'IoT in 3 layers [4]

sistema di dispositivi comunicanti tra loro. Come spesso succede con le tecnologie emergenti ogni marchio, manifacturer e/o sviluppatore non sempre offre compatibilitá con tutti i dispositivi presenti sul mercato. Le ragioni possono essere svariate: per colpa del mancato rispetto degli standard fissati, ma anche per mantenere l'esclusivitá sull'ecosistema, evitando cosí che l'utente acquisti devices della concorrenza in quanto compatibili. Oltre a motivazoni commerciali, tutto l'IoT é caratterizzato da moltissime tipologie di devices difficilmente interconnettibili (e.g. dispositivi poco potenti a batteria che debbono comunicare con nodi piú prestanti) rendendo cosí l'integrazione una operazione complessa.

In questo approccio l'elemento centrale di una rete, il cui scopo é interconnettere le Things di diversa natura, é rappresentato da un Gateway Machine to Machine (M2M). Il Gateway é responsabile dell'adozione dei diversi protocolli per ciascun dispositivo, assicurando un management dell'ecosistema con le funzionalitá di sicurezza. Tra i ruoli del Gateway si possono trovare anche quello di computazione dei dati e di connessione alla rete Internet.

Vengono identificati tre tipi[5] di integrazione tra devices IoT:

- **Integrazione basata sulla rete**
  In questa topologia la rete di sensori wireless (Wireless Sensor Network, abbrevisto WSN) comunica con il Gateway, che é unico e fornisce una panoramica d'insieme delle Thing oltre che l'accesso ad Internet. Nel caso di una rete Mesh multi-hop (come in figura 5.2) si utilizza un rappresentante (Sink) che ha la capacitá di dialogo con il Gateway e con la rete di sensori.

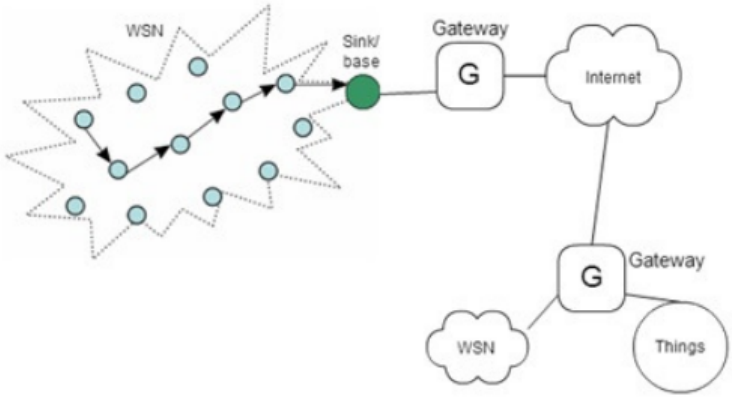

Figura 2.2: Integrazione network-based [5]

- **Integrazione indipendente**
  Nel regime di integrazione indipendente il singolo sensore ha un proprio accesso ad Internet, senza la necessitá di un rappresentante o di un singolo Gateway di riferimento per una intera WSN. Il risultato é la possibilitá di comunicare direttamente con la Thing senza passare per intermediari, come illustrato in figura 5.3 Fornire un indirizzo IP a ciascun dispositivo Internet of Things non é sufficiente per creare una rete di Things mutualmente intercomunicanti[6] Oltretutto non sempre é la soluzione migliore, in quanto per alcuni dispositivi molto semplici ed economici l'ottenimento di un indirizzo IP non é possibile (anche se si tratta di una delle sfide principali dell'Internet of Things)

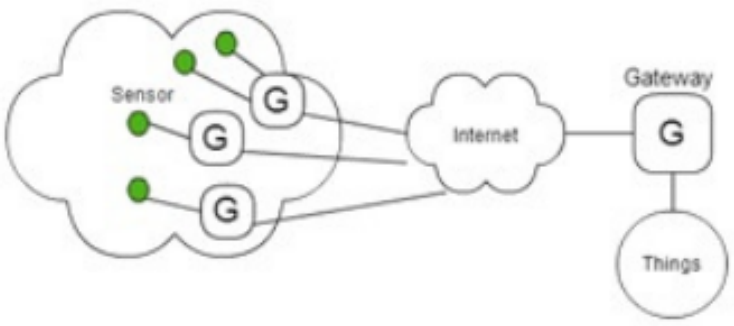

Figura 2.3: Integrazione indipendente [5]

- **Integrazione ibrida**
  L'integrazione ibrida combina le due integrazioni illustrate in precedenza. Si puó trovare una rappresentazione grafica del modello in figura 4.2 Nelle applicazioni reali ogni tipologia di nodo ha caratteristiche energetiche, di costo e di computazione differenti. Non sempre le Things si possono accorpare ad un unico modello di integrazione; si possono peró far coesistere diversi modelli e metterli in comunicazione tra di loro.

L'importanza dell'interoperabilitá tra dispositivi Internet of Things ha interessato il mondo accademico, rendendo necessaria la definizione di una serie di standard di riferimento per la realizzazione di tecnologie compatibili. Nel procedere con la standardizzazzione si é optato per l'utilizzo di tecnologie giá in uso nel Web, come HTTP e WebSocket, piuttosto che definire nuovi protocolli ad-hoc. La motivazione principale si ritrova nel fatto che nella costruzione di protocolli da zero si devono prendere in considerazione una serie di criticitá (come la sicurezza) che sono giá state risolte negli anni con le tecnologie collaudate.

2.2 Web of Things (WoT)

In questo capitolo si tratterá piú nel dettaglio del Web of Things.
Se l'Internet of Things stabilisce le caratteristiche di trasmissione richieste

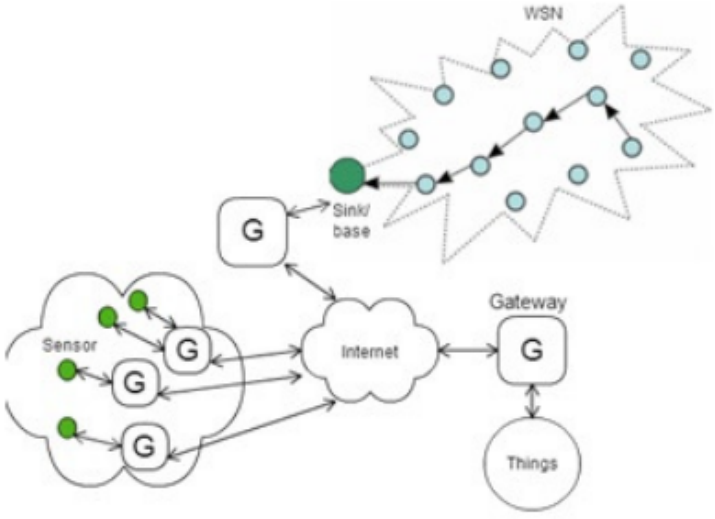

Figura 2.4: Integrazione ibrida [5]

da sistemi in rete, la sfida principale del WoT é quella di integrare le Things nel mondo del Web piuttosto che lasciare queste accedibili attraverso applicazioni Web non integrate[7]. Il vantaggio che si vuole creare é l'utilizzo una Thing come se fosse una qualsiasi risorsa Web. Ció implica che una Thing si possa riutilizzare in svariati contesti, aprendo ad una platea di utilizzi e di potenziali utilizzatori molto superiore. Per molto tempo il mondo accademico si é focalizzato sulla nozione di *risorsa*. Il paradigma delle API REST[7] é risultato fin da subito centrale per rappresentare cambiamenti di stato in una Thing, senza che il server mantenga la sessione ed assicurandosi che le informazioni importanti di uno scenario applicativo siano rappresentati attraverso risorse identificate da URI. Dal momento che le API REST sono largamente utilizzate nel Web, si é scelto di integrarle anche nell'infrastruttura del Web of Things.

Una possibile definizione del WoT é la seguente[8]: *Il Web of Things é una rifinitura dell'Internet of Things, che integra le Things non solo su Internet*

(livello rete), ma anche nell'architettura Web (livello applicativo).
Connettere molte Things ad Internet, fornendo loro un indirizzo IP univoco, permette che loro si scambino informazioni ma non necessariamente che si comprendano. Lo scopo quindi del Web of Things é quello di interporsi come "lingua franca" per una comunicazione efficace tra devices eterogenei.
In prima istanza si discuterá di cosa concettualmente é il Web of Things; successivamente si parlerá delle scelte progettuali intraprese dal W3C per una implementazione standardizzata.

2.2.1 Web of Things tradizionale

Come nell'architettura ISO-OSI, il Web of Things é strutturato in layers.[8] I vari strati in cui si descrive l'architettura WoT non sono da intendersi nel senso stretto del termine ma invece come un'aggiunta di funzionalitá a supporto della tecnologia in ciascun layer.
I layer descritti sono 4+1 (si veda figura 2.5) Al livello piú basso si trova la tecnologia di connessione delle varie Things. Non é da intendersi come un vero layer architetturale in quanto non aggiunge nessuna funzionalitá, ma permette la connessione degli oggetti alla rete. In commercio si trovano diverse tecnologie di interscambio dei dati in quanto la natura delle Things é molto varia e per ciascuna applicazione pratica é piú indicato un metodo di comunicazione piuttosto che un'altro.

Di seguito si descriveranno i layer dell'architettura Web of Things:

- **Access**
  Livello responsabile della trasformazione di una Thing in una Web Thing con cui si puó interagire con protocolli tipici del web, come HTTP/HTML per un interscambio dei dati human-readable.
  Non sempre l'HTTP é la modalitá piú comoda per una comunicazione tra macchine, solitamente si preferiscono protocolli differenti come WebSocket o CoAP e una notazione differente da HTML come XML o JSON-LD.

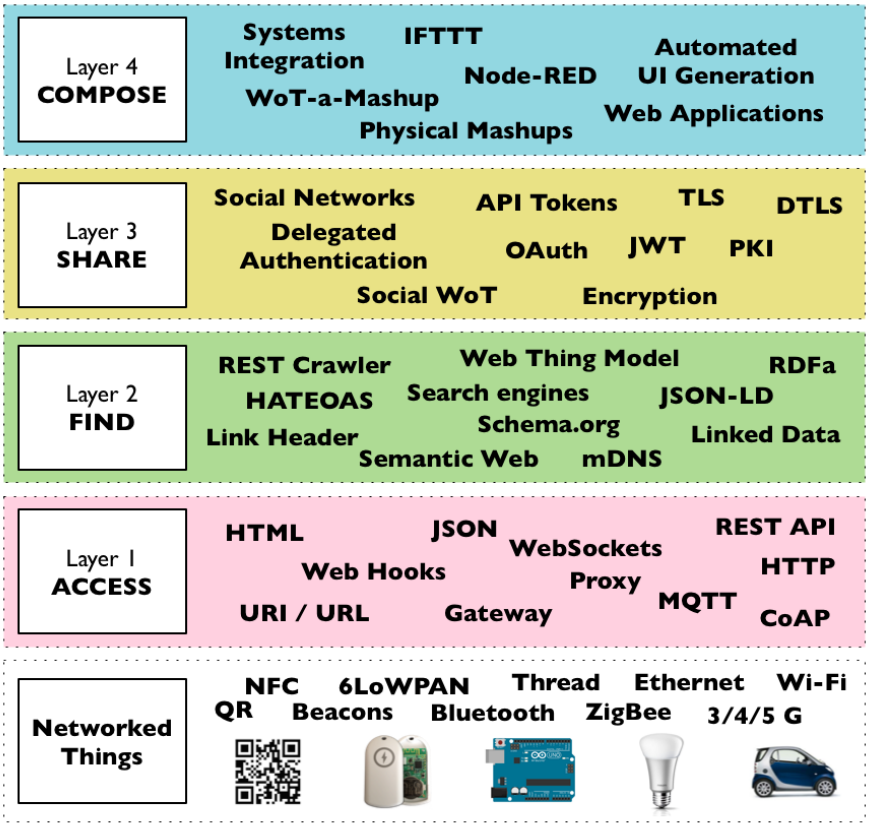

Figura 2.5: L'architettura del WoT composta da 4+1 layers 2.5

- **Find**

  Permette la ricerca da parte di altre applicazioni WoT di funzionalitá della Thing che viene esposta. Viene utilizzato il modello dei dati linkati riusando gli standard semantici giá in uso nel Web.

- **Share**

  Livello responsabile di come i dati generati dalle Things possono essere condivisi e trasmessi su Internet in una maniera efficente e sicura.

- **Compose**

Si pone l'obiettivo di permettere una integrazione tra Things eterogenee in ecosistemi che possono scalare a piacimento.

Il Web of Things, nella sua idea classica, é un protocollo di alto livello disegnato per massimizzare l'interoperabilitá nell'Internet of Things. Si é discusso in svariate sedi su una possibile implementazione reale di tale protocollo. Di seguito si tratterá delle idee progettuali e di design proposte dal consorzio W3C.

2.2.2 W3C Web of Things

Il World Wide Web Consortium (W3C) é una organizzazione operante a livello internazionale che si occupa della definizione di standard e paradigmi universali da utilizzare nel web. Il lavoro del consorzio é stato quello di realizzare uno standard per il WoT per facilitare l'interoperabilitá e l'integrazione marcando la strada per tutti gli sviluppatori e i produttori.
L'idea che c'é dietro al WoT[9] é quella di portare le Things ad un livello astratto e virtuale, cosí da poterle rendere accessibili attraverso le tecnologie che si utilizzano tradizionalmente nel web a prescindere da quali funzionalitá implementa, dalla sua posizione fisica e dai protocolli integrati. Dal momento che lo scopo dello standard WoT é quello di rendere interpretabile una qualsiasi Thing in maniera semplice si é scelto di utilizzare approcci giá esistenti ed affermati nel mondo del web "classico".
Nelle sezioni a seguire si descriverá il Web of Thing allo stato dell'arte attuale[9]; da notare come l'intera documentazione ufficiale sia tuttora in stato di *work in progress*, quindi non immune a cambiamenti in futuro.

Gli elementi architetturali del WoT sono identificati dal W3C come **Building blocks** i quali hanno il ruolo di mostrare le relazioni con gli aspetti architetturali principali di una Thing.
I Building Blocks fondamentali definiti dal W3C sono tre: WoT Thing Description, WoT Binding Templates e WoT Scripting API. In figura 2.6 viene

rappresentata una visualizzazione grafica dei blocchi all'interno dell'architettura di una Thing; una visione piú approfondita delle componenti interne di una Thing viene fornita nella sottosezione 2.2.2.
Da notare come non si possa ignorare il layer di sicurezza, che puó essere considerato come il quarto Building Block.

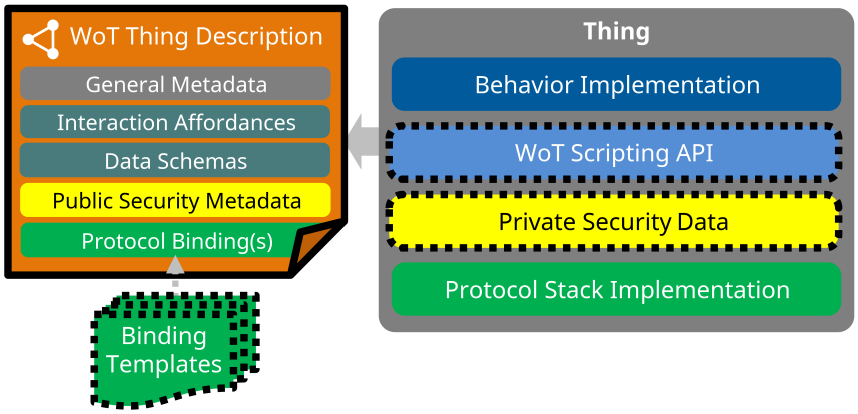

Figura 2.6: Relazioni tra Building Blocks e gli aspetti architetturali di una Thing [9]

I casi d'uso piú comuni vengono illustrati tramite pattern di utilizzo che individuano i ruoli che una Thing puó avere nella rete. Tali pattern sono definiti dall'architettura W3C Web of Things come i seguenti:

- **Controller per thing**
  La Thing viene controllata attraverso la rete da un controller remoto, che puó essere una applicazione da browser oppure nativa

- **Thing to thing**
  la Thing comunica direttamente con un'altra Thing all'interno della rete; questo puó accadere quando un nodo avverte un cambiamento di stato di un sensore, scatenando un'azione in un altro dispositivo nella rete

- **Accesso remoto**
  le funzionalitá della Thing devono poter essere accedibili anche al di fuori della rete *trusted* (nel caso occorre specificare anche layer di sicurezza aggiuntivi nel caso si interagisca dall'esterno, e.g. da uno smartphone connesso alla rete dati)

- **Gateway**
  si tratta di un dispositivo con piú di una interfaccia di rete, una connessa ad Internet e l'altra alla rete locale. Tipicamente l'uso del gateway é indicato quando si richiede di ottenere informazioni dall'esterno, oppure per il controllo a distanza delle Things

- **Edge devices**
  simili ai gateway, ma hanno una certa capacitá computazionale per operare come *bridge* tra vari protocolli di rete.

- **Digital twins**
  si tratta di una rappresentazione virtuale del dispositivo che simula la sua presenza (e.g. quando un dispositivo potrebbe non trovarsi sempre online)

- **Multi-cloud**
  si rende possibile il deploy su diversi dispositivi, a prescindere dal firmware degli stessi, di nuove caratteristiche e funzionalitá per la descrizione della Thing in maniera flessibile

- **Cross-domain collaboration**
  si rendono omogenei ed interoperabili sistemi in domini differenti, che utilizzano tecnologie e protocolli che non necessariamente sono unificati

In figura 2.7 viene illustrata una possibile integrazione tra i vari pattern degli use-case.

Di seguito si descrivono le proprietá richieste dall'architettura.

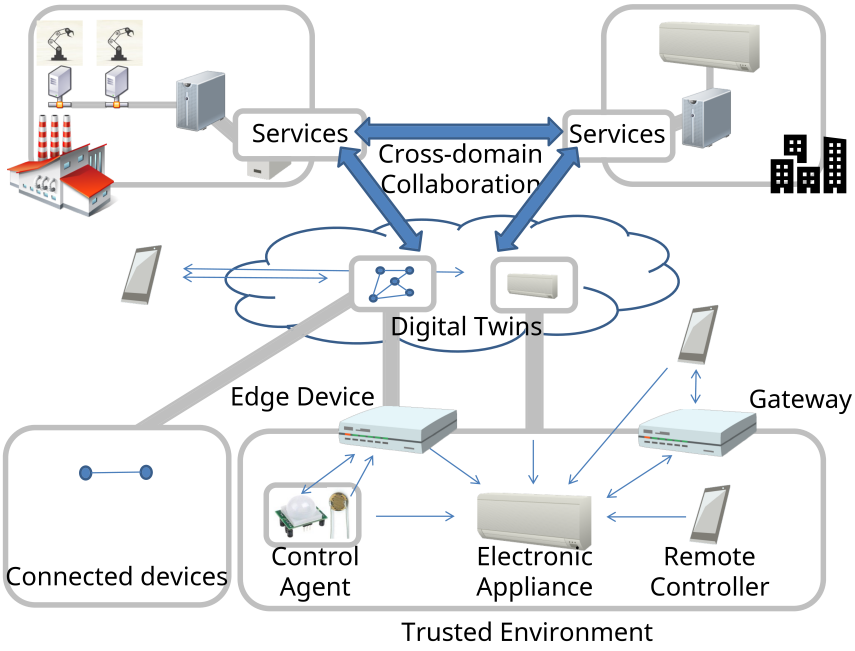

Figura 2.7: Panoramica dei template dei casi d'uso [9]

Architettura W3C WoT

La definizione di **Thing** secondo il W3C WoT[10] é la seguente: *Una Thing é una astrazione di un'entitá fisica oppure virtuale, descritta attraverso metadati da una* ***WoT Thing Description***. Le caratteristiche di una Thing Description verranno meglio spiegate nella sezione 2.2.2.
Uno degli scopi principali del protocollo WoT é quindi quello di descrivere ció che una Thing puó esporre e modificare nel suo ambiente circostante. Il concetto di entitá virtuale deve intendersi come una Thing possa anche essere un'astrazione, e.g. di un insieme di Things (all'interno di una stanza, un appartamento, ecc...) oppure processi remoti di un server. La particolaritá del W3C Web of Things é quella di descrivere quali sono le caratteristiche di una Thing piuttosto che fornire direttive specifiche per l'implementazione; difatti l'intera architettura é formulata in maniera astratta, identificando solamente alcuni requisiti per specifici use-case.

Da precisare come l'intero protocollo WoT non identifica in una thing il ruolo del client e del server, bensí tali ruoli possono essere assunti simultaneamente dallo stesso device. Difatti, ciascuna thing puó essere consumata e puó essa stessa consumare altri dipositivi che implementano il protocollo WoT senza ulteriori intermediari (come illustrato in figura 2.8). Cosí facendo si crea un ecosistema strutturato attraverso una serie di link, che possono fornire informazioni gerarchiche e/o identificare le relazioni tra things.

Figura 2.8: Interazione tra Consumer e Thing

WoT Thing Description (TD)

Le modalitá per il consumo e l'utilizzo di una Thing sono "suggeriti" dal protocollo W3C stesso, senza fornire uno scheletro rigido. Il sistema di rappresentazione del WoT é la Thing Description **(TD)** che fornisce una rappresentazione auto-descrittiva di una Thing attraverso il formato JSON-LD. La TD rappresenta dati in una struttura semantica, includendo il contesto in cui la Thing opera.
La TD (che puó essere considerato come l'indice della Thing[11]) contiene le informazioni utili sull'interfaccia, come elementi testuali (nome, ID, ecc...), link per rappresentare relazioni, protocolli di sicurezza utilizzati. Le modalitá con le quali la TD descrive le interazioni con la Thing sono chiamate **Interaction Affordance**.
Il concetto di Affordance viene definito come: *"le proprietá fondamentali che determinano come un'oggetto possa essere utilizzato"* ([10], cap. 6.2). Nella Thing Description si possono trovare gli endpoint (URI) per ciascuna In-

teraction Affordance. Nell'architettura WoT si riconoscono tre tipologie di Interaction Affordances:

- **Proprietá**
  sono variabili che corrispondono a stati interni della Thing (e.g. la temperatura rilevata dal termometro) oppure parametri di configurazione. Una proprietá dev'essere necessariamente accedibile in lettura e opzionalmente anche in scrittura (attraverso la chiamata ad un'azione)

- **Azioni**
  sono funzioni della Thing, che possono modificare lo stato interno della Thing oppure far scattare alcune azioni (e.g. chiusura di una serratura elettronica).

- **Eventi**
  é una condizione alla quale il Consumer puó mettersi in ascolto (solitamente attraverso una procedura di sottoscrizione, specificata dalla TD) in attesa di un cambio di stato interno alla Thing (e.g. al superamento di una soglia di temperatura).

WoT Binding templates

Il mondo IoT non ha un protocollo di accesso al dispositivo unificato, dal momento che ciascun protocollo, con le sue peculiaritá, puó essere piú o meno indicato per una certa applicazione. Con il WoT non si cerca quindi un protocollo universale, piuttosto una metodologia di accesso universale che possa incrociare la moltitudine di possibilitá che si incontrano nell'IoT. I Binding Templates forniscono le informazioni necessarie attraverso metadati (contenuti nella Thing Description) per descrivere come una Thing comunica e quali sono i protocolli utilizzati dalla stessa a seconda delle Interaction Affordance che si intende utilizzare.
Il Consumer deve quindi implementare il Protocol Binding con uno stack per la comunicazione con il protocollo specificato dal Binding Template che

si intende utilizzare. Gli elementi che caratterizzano un Binding Template sono i seguenti:

- **IoT Platform**
  si descrivono modifiche proprietarie al layer di applicazione del protocollo di comunicazione, come un header HTTP differente

- **Media-type**
  le varie piattaforme IoT possono avere un loro formato di rappresentazione dei dati scambiati; il media-type lo identifica univocamente

- **Transfer protocol**
  questo campo identifica il protocollo di comunicazione dei dati per il Binding Template. Possono essere numerosi e dipendono dal tipo di applicazione che si vuole implementare; un esempio possono essere i protocolli HTTP, CoAP oppure WebSocket

- **Subprotocol**
  i transfer protocol possono supportare estensioni, le quali se vengono utilizzate debbono essere necessariamente dichiarate. Un esempio é il sottoprotocollo LongPoll per HTTP

- **Security**
  ogni Binding Template puó supportare un differente layer di sicurezza, come DTLS, OAuth o IPSec.

WoT Scripting API

Le WoT Scripting API sono un building block opzionale definito dal W3C, che permette la definizione di API compliant alle specifiche ECMAScript. Sono implementate all'interno di una Thing e ne definiscono il comportamento della stessa oppure quello di consumers oppure intermediari con cui la Thing comunica. Le Scripting API sono incluse in script chiamate dal sistema runtime, permettendo cosí di poter riutilizzare tali porzioni di codice da altri sistemi IoT, riducendo i costi di integrazione ed aumentando la produttivitá.

Le specifiche delle Scripting API definiscono le interfacce di programmazione per permettere il discover, il fetch, la produzione, l'esposizione ed il consumo delle Thing Description. Il runtime istanzia gli oggetti necessari rappresentanti le varie Interaction Affordance di altre Things.

Linee guida su privacy e sicurezza nel WoT

Il consorzio W3C fornisce le linee guida sulla sicurezza[12]. Non vengono imposte strutture specifiche, in quanto la sicurezza é un aspetto trasversale che si applica a tutti i building blocks e a tutte le porzioni sensibili in materia di privacy nella trasmissione dei dati. Il documento sulla privacy e sulla sicurezza indica l'utilizzo di meccanismi giá esistenti nel Web per rendere sicuro il mantenimento e l'invio dei dati, identificando le criticitá in tre punti:

- **Disegno della Thing Description**
  La Thing Description deve evitare di esporre informazioni della Thing (e.g. versione firmware), limitare l'accesso a client autorizzati, non permettere l'alterazione di campi interni e/o protetti
- **Dati sensibili del sistema**
  L'invio di dati sensibili (specialmente quelli riguardanti persone fisiche) debbono essere trasferiti con protocolli protetti (come HTTPS o CoAPS)
- **Disegno delle interfacce WoT**
  Le interfacce di rete devono permettere le funzionalitá essenziali, utilizzando schemi di controllo dell'accesso ed evitando processi pesanti senza l'autorizzazione del Consumer

WoT Servient

Nelle sezioni precedenti si é illustrato cosa sono i building blocks; un WoT Servient é una implementazione dello stack software che integra i WoT Building Blocks. I Servient possono esporre e/o consumare Things, assumendo

entrambi i ruoli di client e server. I componenti di un WoT Servient sono quindi il codice che costituisce la logica applicativa della Thing, la Thing Description e la descrizione fornita da quest'ultima delle Interaction Affordance (con le quali si puó interagire attraverso le Scripting API e i Protocol Bindings)

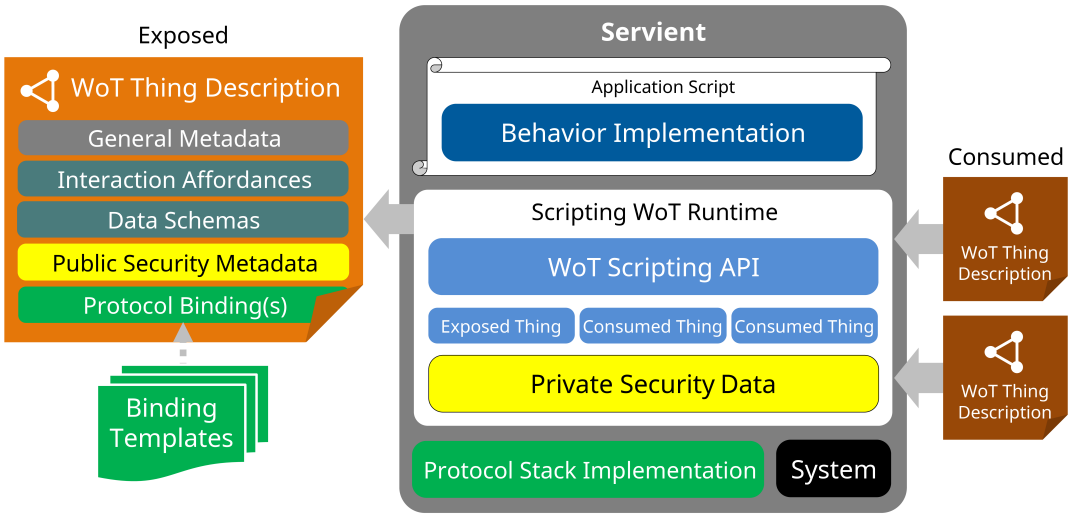

Figura 2.9: Architettura di un WoT Servient con le Scripting API

Di seguito si indicheranno i ruoli di ciascun componente del WoT Servient mostrato in figura 2.9.

- **Behaviour Implementation**
  Il comportamento della Thing ne definisce la logica operazionale. In questo blocco, costituito da diverse funzioni e script, viene descritto come il dispositivo debba analizzare i dati in input (e.g. da un sensore), gestire le Interaction Affordance (e.g definendo la funzione da invocare alla chiamata di un endpoint) e definendo il proprio compito nel ruolo di Consumer o Intermediary.

- **Scripting WoT Runtime**
  Il Runtime permette di mantenere attivo l'ambiente di esecuzione per

la logica comportamentale del device. Si occupa di ottenere, processare e serializzare le Thing Description, gestire gli endpoint delle Interaction Affordance attraverso i Binding Templates (i quali anch'essi vengono eseguiti dal Runtime) esponendo le interfacce basandosi sulle Scripting API definite. Qui vengono gestiti i dati che il device rileva oppure processa; per questo motivo il Runtime solitamente é posto in un ambiente protetto da accessi esterni.

- **Protocol Stack Implementation**
  L'insieme dei protocolli di un Servient implementano l'interfaccia della Thing esposta, utilizzabile dai Consumers per interagire con il dispositivo. Nei casi in cui si utilizzano protocolli di comunicazione standard, i Protocol Stacks producono messaggi specifici per piattaforma (e.g. per comunicazioni HTTP si utilizzerá un "dialetto" differente rispetto a CoAP). La Thing Description espone i meta-dati per la configurazione dello stack di altri devices, cosí da rendere possibile l'intercomunicazione.

- **System API**
  Sotto questo punto si possono identificare le interfacce hardware fornite dalla Thing con le quali il WoT Servient ha il compito di interagire. Non sempre il Servient e l'hardware della Thing sono sullo stesso dispositivo ma anzi debbono anch'essi dialogare attraverso protocolli proprietari non contemplati dallo stack WoT. Le System API (realizzate dal manifacturer dell'hardware) permettono tale scambio di dati. In quanto la natura dell'hardware dei dispositivi é intrinsecamente eterogenea, non vengono definiti standard per non limitarne la scalabilitá e la portabilitá.

Capitolo 3

Micro-WoTServient

In questo capitolo si tratterá del progetto Micro-WoTServient. Il progetto é un ulteriore sviluppo dello stack embeddedWoTServient presentato da Daniele Rossi nella sua Tesi Magistrale[13].

3.1 Obiettivi

Il progetto consiste nel progettare, implementare e validare un WoT Servient per dispositivi embedded seguendo le specifiche fornite dal consorzio W3C, descritte nella sezione *W3C Web of Things*. In particolare la piattaforma é stata sviluppata per realizzare codice funzionante su board Espressif ESP32, utilizzando gli strumenti forniti dal compilatore di Arduino. Il lavoro é stato pensato per portare agilmente tale software anche su altre board prototipali programmabili con Arduino (che presentino caratteristiche di computabilitá e connettivitá compatibili con ESP32).
Il progetto prevede anche una GUI che permetta la progettazione della Thing Description che deve essere esposta dalla Thing, includendo porzioni di codice personalizzate a seconda delle esigenze, offrendo la possibilitá di aggiungere liberie esterne, variabili globali e funzioni. Dal momento che la natura della Thing che l'utente realizza é intrinsecamente dipendente dall'ambito di utilizzo, l'applicazione permette la scelta dei Binding Templates per ciascuna

delle sezioni della Thing Description. I Binding Templates che si é scelto di implementare sono HTTP (con l'uso del subprotocol HTTP LongPoll per il polling degli eventi), WebSocket e CoAP. Il design del programma é stato articolato in modo da poter integrare Binding Templates aggiuntivi senza apportare modifiche al core applicativo.
In seguito lo scrpit produrrá, attraverso le informazioni fornite dall'utente, la Thing Description in formato JSON ed il codice Processing da *flashare* sul dispositivo che comprenda tutto lo stack WoT. Una volta prodotto il codice del Servient l'applicazione permetterá di caricarlo sulla board prototipale attraverso i tool messi a disposizione da Arduino. Nelle sezioni seguenti si utilizzerá il termine *sketch* per riferirsi al file contenente la logica per il microcontrollore prodotto automaticamente dal programma.

3.2 Requisiti

Ora si andranno ad analizzare i requisiti che l'applicazione micro-WoTServient deve rispettare.

3.2.1 Requisiti funzionali

I requisiti funzionali del WoT Servient sono definiti come i seguenti:

- **Creazione della Thing Description**
  L'utente deve fornire tutti i dati riguardanti la Thing che si vuole generare. Il programma deve produrre una Thing Description compliant allo standard W3C Web of Things che dovrá poi essere esposta dal WoT Servient.

- **Scelta dei Binding Templates**
  L'utente, per ciascuna Interaction Affordance e per l'esposizione della Thing Description, deve poter effettuare una scelta su uno o piú Binding Templates messi a disposizione dall'applicazione

- **Creazione sketch Arduino**
  Il programma in seguito realizza in maniera automatizzata un file di scripting in linguaggio Processing compatibile con la board embedded specificata. Il file di scripting contiene la logica applicativa che deve essere eseguita dalla Thing

- **Compilazione e flashing dello sketch**
  L'utente puó compilare il codice prodotto dall'applicazione, verificando cosí la correttezza sintattica dello sketch. In seguito l'utente puó scegliere di *flashare* il firmware appena prodotto sul microcontrollore

3.2.2 Requisiti tecnici

I requisiti tecnici richiesti dal WoT Servient sono i seguenti:

- **Esposizione Thing Description**
  La Thing Description deve poter essere visualizzata da altri dispositivi. Si richiede quindi che la Thing si possa connettere alla rete, avendo cosí un URL/indirizzo IP in modo da poter essere raggiungibile e consumabile.

- **Protocol Bindings**
  Dal momento che una Thing puó avere diversi metodi di interazione utilizzando le tecnologie del Web, lo sketch prodotto deve poter implementare i Binding Templates necessari alla produzione e al consumo. Nel dettaglio, la Thing deve poter comunicare attraverso i protocolli HTTP, WebSocket e CoAP. Il modello per lo scambio dei dati si basa sul paradigma delle API REST.

- **Codice ottimizzato**
  Il codice prodotto dal programma é inteso per essere eseguito su piattaforme che non hanno livelli alti di prestazione. L'overhead prodotto dal framework del Servient deve essere ridotto al minimo per dare piú

spazio di manovra possibile all'utente nell'inserimento di codice e/o librerie personalilzzate.

3.3 Architettura

In questa sezione si discuterá l'architettura del progetto micro-WoTServient. In figura 3.1 viene illustrato il flusso di utilizzo dell'applicazione.

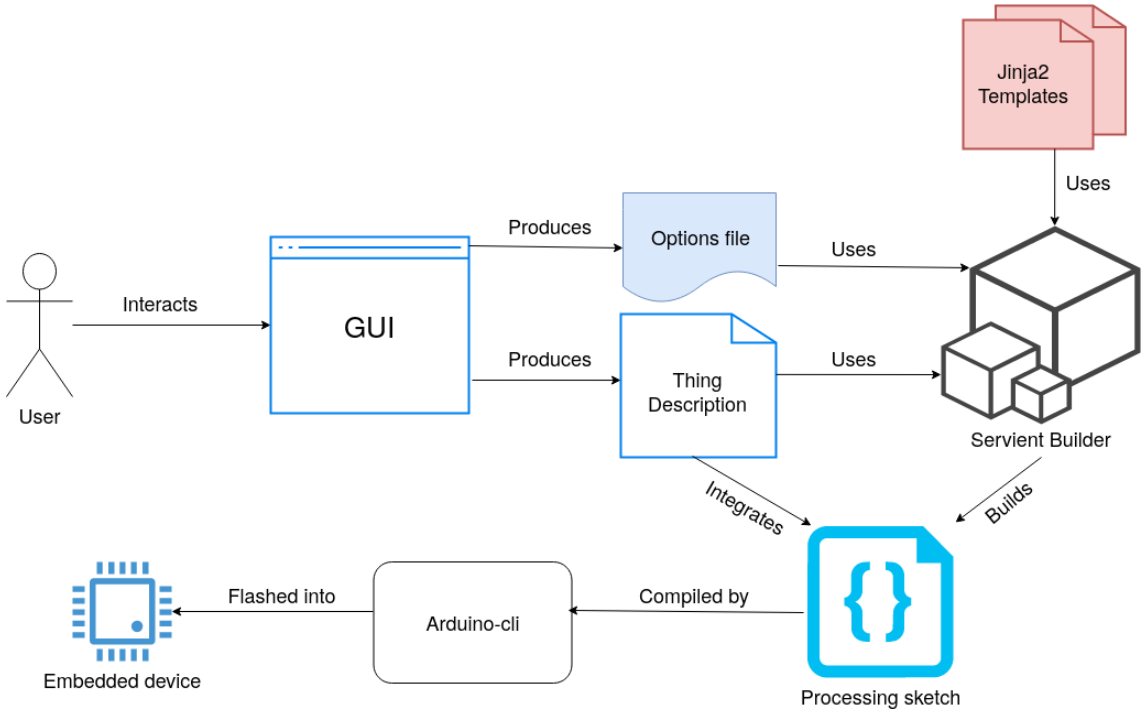

Figura 3.1: Architettura micro-WoTServient

Di seguito si descriveranno piú nel dettaglio le componenti dell'architettura:

3.3.1 GUI

É l'unico entry point del programma e l'unico componente che l'utente puó utilizzare. Tramite tale interfaccia viene permesso all'utente il compi-

mento delle seguenti interazioni:

- inserire i dati sulla Thing (nome, URI, versione, ecc...)
- inserire i dati per la connessione alla rete WiFi, come SSID ed eventualmente password
- inserire le porte a cui i Binding Templates dovranno rimanere in ascolto
- inserire variabili globali, costanti, includere librerie, definire macro
- inserire funzioni o porzioni di codice addizionale
- per ogni Interaction Affordance, definire un insieme di Binding Templates che rispondano a tali endpoint
- inserire le Properties, indicando nome e tipo
- inserire le Actions, indicando il nome, numero e tipo di parametri, tipo di output e funzione da invocare alla chiamata dell'endpoint
- inserire gli Events, indicando nome, schemi di sottoscrizione e cancellazione (per i Binding Templates che lo permettono), condizione per la quale l'evento verrá triggerato
- *buildare* il progetto da compilare (creazione dello sketch)
- compilare e *flashare* lo sketch su una specifica board

Gli output che produce la GUI sono il file della Thing Description e il file delle opzioni. Una volta compilati tutti i form con i dettagli implementativi della Thing, la GUI chiamerá il Servient Builder passando in input i files prodotti in precedenza,

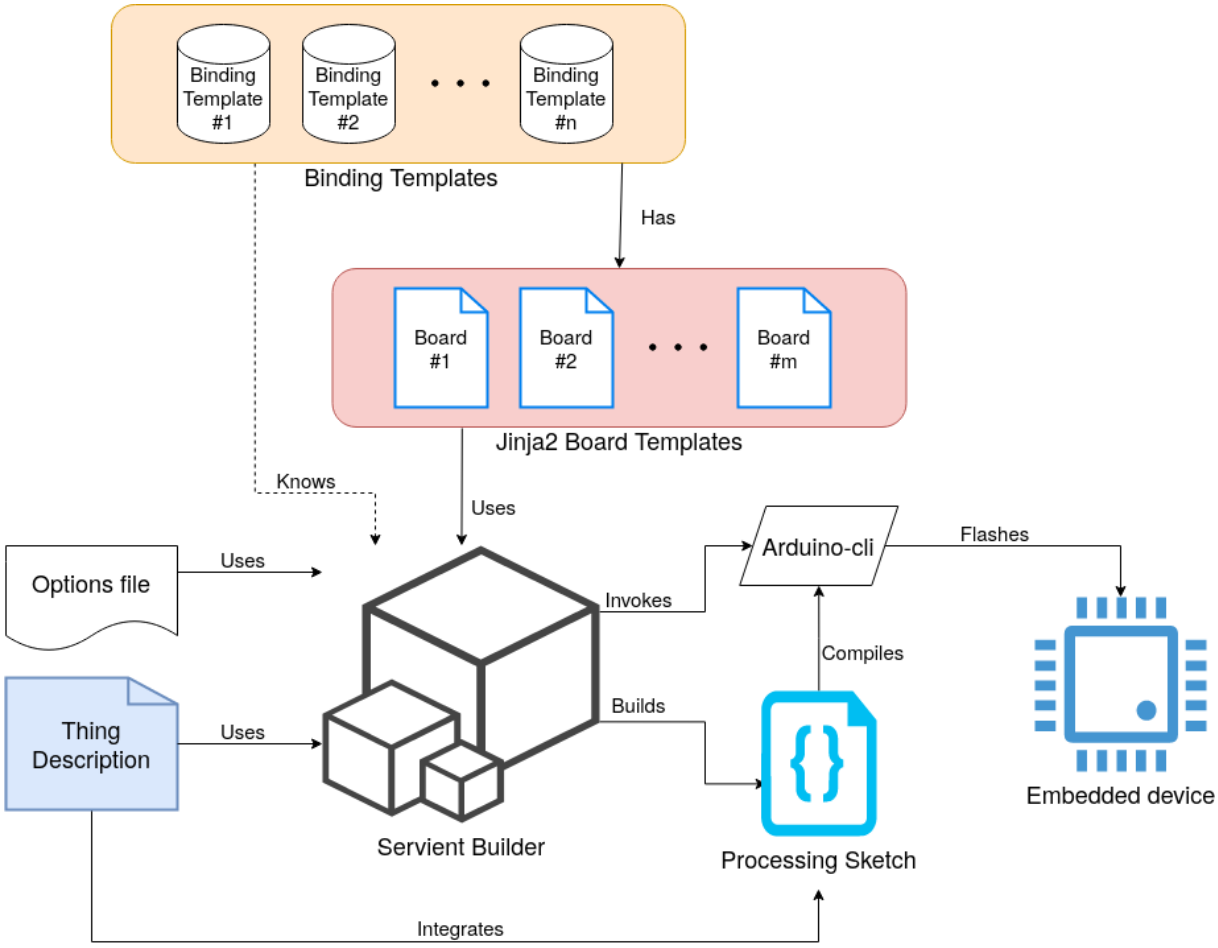

Figura 3.2: Architettura Servient Builder

3.3.2 Servient Builder

Il Servient Builder é il core del programma. Richiede in input un file di template specifico per il microcontrollore e i file della Thing Description e delle opzioni.

I compiti che assolve il Servient Builder sono i seguenti:

- download delle librerie dei Binding Templates (se non presenti)
- download ed installazione dei driver del microcontrollore (se non presenti)

- creazione dello sketch in linguaggio Processing
- compilazione e upload dello sketch (avvalendosi del tool Arduino-cli)

Nella sua logica é presente un Jinja2 Template Engine che utilizzando i template per le board da programmare genera lo sketch a seconda del file di opzioni e della Thing Description generati dalla GUI.
Nei templates sono presenti le porzioni di codice necessarie per includere la definizione degli oggetti dei Binding Templates. Il core si occuperá di *buildare* il file di scripting a seconda delle direttive contenute nel template (e quindi del tipo di microcontrollore) scelto per la compilazione.
Da notare come il modulo intero non sia direttamente implicato nella costruzione del codice dei Binding Templates. Questo perché si é voluto svincolare la modifica del core applicativo nel caso si voglia in futuro aggiungere protocolli specifici. Il Servient Builder é unicamente a conoscienza di quali sono i link alle repository GitHub dei Binding Templates per scaricare le librerie nel caso non siano integrate nel path di Arduino.
In output viene prodotto lo sketch che dovrá essere compilato e caricato sulla board. Tale compito viene assolto dal tool Arduino-cli, che fornisce il compilatore del linguaggio Processing e permette il flashing di un dispositivo compatibile.

3.3.3 Binding Templates

Per questo progetto si é scelto di integrare i Binding Templates come librerie perfezionate ad-hoc per tale scopo. Allo stato attuale ne ho sviluppati tre, e permettono alla Thing di dialogare (a discrezione dell'utente) i protocolli HTTP, WebSocket e CoAP. All'interno dello sketch verranno incluse solamente le librerie (e quindi i Binding Templates) che l'utente intende integrare.
Nulla vieta, come spiegato in precedenza, di incrementare il numero di Binding Templates integrabili all'interno del device. Verrá discusso approfon-

ditamente il metodo con il quale si é scelto di implementare tali librerie in maniera flessibile nel paragrafo 4.1.3.

3.4 Scelte progettuali

Come board per testare lo stack WoT-Servient é stata utilizzata la ESP32 prodotta da Espressif[1](in figura 5.2). La decisione su tale devices é ricaduta in quanto fornisce una buona potenza di calcolo (seppur limitata, ma superiore ad altri dispositivi piú blasonati della stessa fascia come Arduino UNO, Atmel ATMega328), connettivitá Wi-Fi e BLE (Bluetooth Low-Energy), dispendio energetico limitato, costo e dimensioni contenute. Tale scelta ha vincolato nell'uso di librerie create ad-hoc per permettere la compatibilitá con tale dispositivo.
In precedenza per progetti simili si utilizzavano dispositivi differenti (come NodeMCU ESP8266); la decisione di spostarsi su ESP32 é dovuta dalla disponibilitá di un NIC BLE integrato nel SoaC (System-on-a-Chip) che NodeMCU non possiede.

Il codice *buildato* dal Template Engine é codice Processing, compatibile con Arduino IDE (e quindi con ESP32).
I Protocol Binding integrabili nella logica applicativa della Thing sono tre: HTTP (con subprotocol HTTP LongPoll per l'emissione di Events), WebSocket e CoAP. Sta all'utente la decisione sull'utilizzo dei Protocol Bindings piú adatti per ciascuna Interaction Affordance definita nella Thing Description. Le Scripting API integrate nello stack del Servient sono API REST.

[1]https://www.espressif.com/en/products/socs/esp32

Figura 3.3: Microcontrollore Espressif ESP32

Capitolo 4

Implementazione

La realizzazione di tale progetto, trattandosi di un sistema composto da varie componenti tra loro eterogenee, ha obbligato l'uso di diverse tecnologie e alla loro integrazione. La repository del progetto micro-WoTServient é raggiungibile tramite il seguente link:
`https://github.com/Chello/WoT-microcontroller-servient-GUI`.
Nelle sezioni a seguire si descriveranno le tecniche utilizzate per raggiungere al risultato.

4.1 Tecnologie

A seconda dell'ambito di appartenenza di ciascun elemento costituente il progetto intero, si é optato per l'uso di svariate tecnologie.
La GUI é stata scritta con il framework grafico Electron (supportato da JavaScript).

4.1.1 JavaScript ed Electron

Per l'interfaccia grafica (GUI) é stato utilizzato il framework Electron (versione 8.3.x) eseguito da Node V10.21.x, scritta con il ilnguaggio JavaScript. La scelta é ricaduta su tale supporto per svariati motivi

- Per il disegno della GUI, Electron utilizza l'engine del browser Chromium per la visualizzazione dell'interfaccia. Questo implica l'utilizzo del linguaggio a markup HTML in accoppiata con fogli di stile CSS per la formattazione dell'applicazione. Tale approccio semplifica la realizzazione della GUI, riducendosi alla creazione di un simil-website funzionante standalone.

- Per la realizzazione della Thing Description in formato JSON-LD é stato utile l'utilizzo della libreria Javascript json-editor[1], in quanto costruisce in modo automatizzato un form complesso a piacimento sulla base di uno schema definito da un file json-schema. Tale libreria é stata utile anche per la realizzazione del form delle opzioni di building che andranno poi passate al core applicativo.

Per il raggiungimento dei fini del progetto si é reso utile includere nel file *package.json* (con le direttive delle componenti da scaricare dalla repository di NPM) le seguenti librerie:

- **jszip**[2]
  Utile per la realizzazione dei file di salvataggio. Ciascun salvataggio consiste in un file compresso in formato zip (rinominato in .wotsrv) che include al suo interno i documenti JSON della Thing Description e delle Building Options

- **ace-builds**[3]
  Code editor scritto in JavaScript. Utile per la visualizzazione delle *textarea* che devono contenere porzioni di codice

- **xterm**[4]
  Componente front-end per la realizzazione di un terminale che imple-

[1] https://github.com/json-editor/json-editor
[2] https://www.npmjs.com/package/jszip
[3] https://www.npmjs.com/package/ace-builds
[4] https://www.npmjs.com/package/xterm

menti i canali stdin, stdout e stderr per la visione verbosa dei messaggi provenienti dal Servient Builder, del compilatore e dell'uploader.

- **jquery**[5]
  Popolare framework JavaScript per una interazione piú semplice ed agevole con il DOM

- **bootstrap**[6] **/ font-awesome**[7]
  Framework CSS, utili per il disegno di una grafica accattivante e l'inserimento di icone

4.1.2 Python, Jinja2

Il core applicativo consiste in uno script in Python v.3.6.x[8]. La scelta di tale linguaggio di programmazione é ricaduta grazie alla sua semplicitá sintattica, disponibilitá di librerie e di documentazione online, potenza e velocitá di esecuzione. Di seguito si elencheranno alcune tra le proprietá caratterizzanti del linguaggio:

- é Object Oriented (ma non impone strettamente l'uso e la creazione di oggetti e/o classi)

- é debolmente tipizzato (il controllo dei tipi é effettuato a runtime)

- é interpretato. Se si esegue l'interprete senza fornire in input alcun programma si accederá alla modalitá interattiva, particolarmente utile in fase di sviluppo per testare porzioni di codice

- lo stile di programmazione é parte integrante della sintassi del linguaggio a favore della leggibilitá

- dispone di un Garbage Collector per gli oggetti istanziati di cui non si ha piú riferimenti

[5]https://www.npmjs.com/package/jquery
[6]https://www.npmjs.com/package/bootstrap
[7]https://www.npmjs.com/package/font-awesome
[8]https://www.python.org/

Per il build del programma in Processing compatibile con Arduino (e quindi con le board prototipali messe a disposizione da tale tecnologia) si é utilizzato un template scritto in Jinja2[9]. Nonostante tali linguaggi di templating sono spesso utilizzati in ambito Web (ad esempio per la realizzazione a runtime di pagine HTML), Jinja2 é risultato particolarmente efficace anche nella realizzazione di sketch. Un file di template rappresenta una tipologia di board. Per ora l'unico file di template sviluppato é quello riguardante sketch compatibili con Espressif ESP32, ma si possono aggiungere altri template che realizzano il codice per altri microcontrollori. Un Template scritto in Jinja2 ricopia la sua parte statica nell'output e a seconda dell'oggetto dato in input computa e riscrive la parte dinamica, che viene codificata come segue:

- `{% ... %}` per le istruzioni e le dichiarazioni
- `{{ ... }}` per espressioni di echo e di stampa in output
- `{# ... #}` per commenti che non debbono essere ricopiati in output

Con la codifica precedentemente illustrata si possono descrivere programmi scritti in Processing con una buona flessibilitá.
Il progetto prevede anche la fase di compilazione e caricamento su board prototipale del codice realizzato. Per il raggiungimento di tale risultato si utilizza il tool a riga di comando Arduino-cli, dove se ne discuterá meglio nella sottosezione 4.1.3. Dal momento che Arduino-cli é un comando bash, si é utilizzata la libreria Python **subprocess** per eseguire comandi nei path del sistema operativo.

4.1.3 Processing e Arduino

Processing[10] é il linguaggio di programmazione che viene utilizzato per il coding di microcontrollori prototipali compatibili con Arduino IDE. Si é

[9] https://jinja.palletsprojects.com/en/2.10.x/templates/
[10] https://processing.org/

optato per sviluppare con questo linguaggio in quanto risulta essere quello comunemente piú scelto per tali utilizzi.

Di seguito si elencheranno le caratteristiche peculiari del linguaggio:

- é un linguaggio di tipo C-Like, ereditando molta della sintassi di quest'ultimo
- implementa il paradigma di programmazione ad oggetti presente in C++, senza peró imporre l'uso di classi e oggetti (anche se fortemente consigliato)
- gli sketch debbono essere compilati in codice macchina prima di poter essere eseguiti
- `setup()` é la prima funzione che viene chiamata all'inizio dell'esecuzione dello sketch. Funge da entry point e sostituisce l'uso della funzione `main()` tipica di C/C++
- `loop()` é la funzione che verrá eseguita ciclicamente al termine della funzione `setup()`

Gli sketch prodotti debbono essere successivamente compilati per un certo microcontrollore, e di seguito flashati all'interno di essi. Questi due ruoli vengono assolti solitamente dall'Arduino IDE, interfaccia GUI con un editor utile per la programmazione, la compilazione e il caricamento su board prototipale del codice macchina. Arduino ha sviluppato un'interfaccia a riga di comando funzionalmente identica alla controparte GUI, chiamata Arduino-cli. Le funzionalitá disponibili sono quelle per compilare uno sketch, caricarlo su un microcontrollore, ricercare e scaricare driver, board e librerie dal repository ufficiale di Arduino.

Nel Web of Things il tipo di meta-dato piú utilizzato per la trasmissione di messaggi é il JavaScript Object Notation (JSON). Nonostante sia il metodo generalmente preferito per la sua comprensibilitá e semplicitá, il JSON non é nativamente supportato da Processing. Per aggiungere tale supporto si

utilizza la libreria ArduinoJSON[11], ampliamente documentata, supportata e di semplice utilizzo. Fornisce la struttura dati per creare, accedere e modificare Array e Object che possono essere serializzati/deserializzati in formato JSON. Nell'intero sketch autogenerato si utilizzano le istanze fornite dalla libreria ArduinoJSON in sostituzione agli array "built-in", in quanto risultano piú comodi e serializzabili per l'invio di dati. L'utilizzo di oggetti e array JSON é una funzinonalitá richiesta da tutto lo stack ed é utilizzata trasversalmente da ogni modulo.

Librerie Per rendere il progetto modulare si sono realizzate delle librerie che implementano i Binding Templates che l'utente specifica per ciascuna Interaction Affordance e per la stessa esposizione della Thing Description. Nella genererazione automatica dello sketch vengono scaricate ed incluse a seconda del loro utilizzo. Le librerie realizzate sono tre, una per Binding Template.

- `embeddedWoT_HTTP_LongPoll`
  Implementa HTTP (utilizzando il subprotocol longpoll per l'ascolto di eventi) avvalendosi della libreria `ESPAsyncWebServer`[12], che permette di mantere aperti simultaneamente fino ad 8 sessioni HTTP. Lo scopo di questo Binding Template é quello di fornire un metodo semplice di visualizzazione ed invio dei dati del microcontrollore, utile in quelle occasioni nelle quali risulta comoda una interfaccia con un comune Web Browser (ad esempio in fase di debug durante la realizzazione di una Thing).

- `embeddedWoT_WebSocket`
  Implementa il protocollo WebSocket utilizzando la libreria `WebSocketsServer`[13]. Il Binding Template usa il protocollo di livello trasporto TCP connection-

[11] https://arduinojson.org/
[12] https://github.com/me-no-dev/ESPAsyncWebServer
[13] https://github.com/Links2004/arduinoWebSockets

oriented, per rendere la comunicazione affidabile e persistente (a scapito delle prestazioni di rete).

- `embeddedWoT_CoAP`
  Implementa il protocollo CoAP utilizzando la libreria `coap-simple`[14]. Questo Binding Template é stato disegnato per essere leggero, infatti si é scelto il protocollo di livello trasporto UDP (connectionless) per aggiungere il minimo overhead possibile.

Le librerie che forniscono i Binding Templates sono state disegnate in maniera simile per omogeneizzare l'utilizzo. Alcuni metodi in comune sono i seguenti:

- `exposeProperties()` permette l'esposizione di proprietá della Thing. Viene richiesto un array di endpoints REST corrispondente ad un array di callbacks da lanciare al momento della chiamata dell'Interaction Affordance

- `exposeAction()` permette l'esposizione di azioni della Thing. Il funzionamento di tale metodo é il medesimo illustrato per la funzione `exposeProperties()`

- `exposeEvents()` permette l'esposizione di eventi della Thing. Il metodo richiede gli endpoints a cui un Consumer si puó mettere in ascolto di un evento.

- `sendTXT()` permette l'invio di eventi. Viene richiesta una stringa testuale con il corpo del contenuto da trasmettere e il nome dell'evento a cui bisogna indirizzare l'invio. Tutti i Consumer che si sono in precedenza collegati a tale evento verranno notificati.

- `bindEventSchema()` permette di specificare uno schema di sottoscrizione o cancellazione per gli eventi (tale metodo non é disponibile per il binding HTTP LongPoll).

[14] https://github.com/hirotakaster/CoAP-simple-library

4.2 Componenti realizzate

Di seguito si descriveranno le componenti realizzate all'interno del progetto.

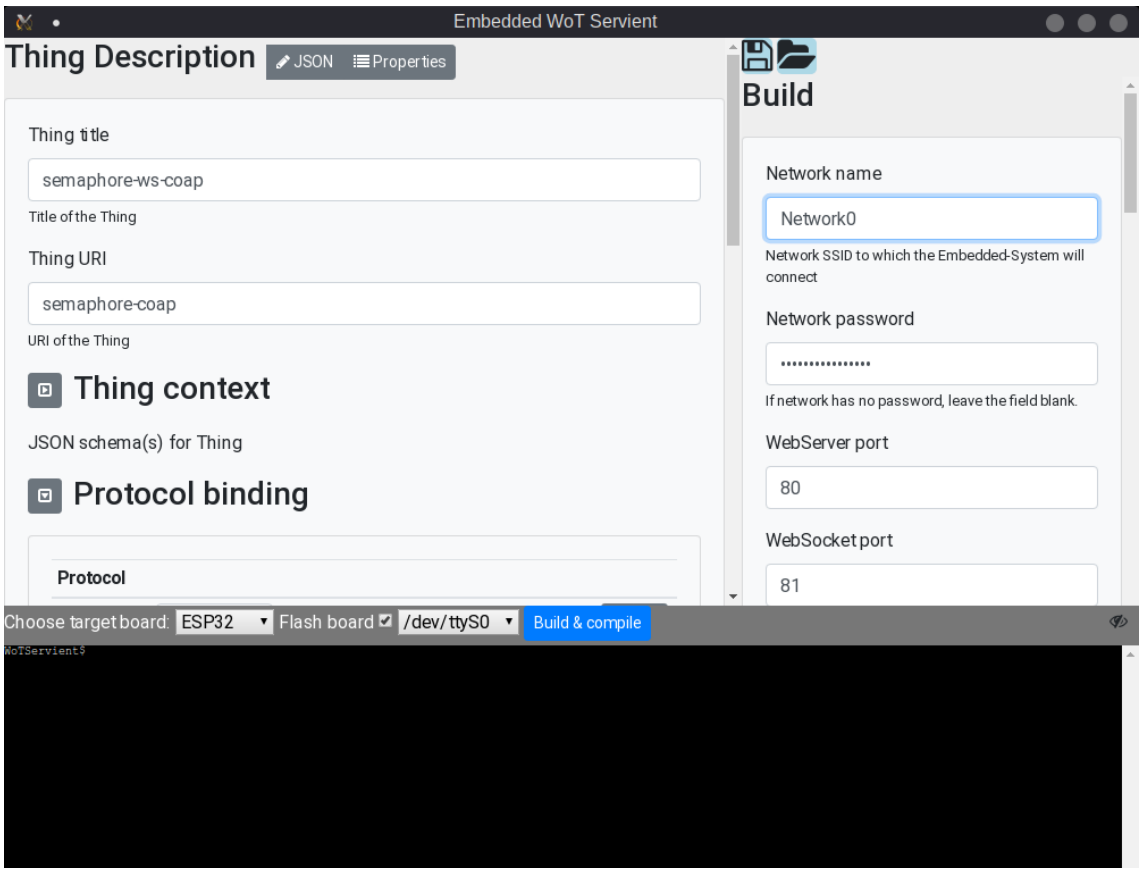

Figura 4.1: EmbeddedWoTServient GUI

GUI (Interfaccia Grafica) Il componente con il quale l'utente si interfaccia é la GUI che si presenta come in figura 5.3. Si possono identificare tre elementi nell'interfaccia grafica:

- **Thing Description** (colonna di sinistra)
  In questa sezione vengono richieste le caratteristiche della Thing Description. Le informazioni richieste sono le seguenti:

  - Meta-dati che caratterizzano la Thing (nome, URI, data di creazione, ecc...)
  - Properties da esporre, scegliendo nome e tipo
  - Actions da esporre, compilando nome, numero e tipo di input, tipo di output, corpo della funzione da invocare
  - Events da esporre, compilando nome, schemi di sottoscrizione e cancellazione, azione che triggera l'evento, condizione per la quale l'evento viene invocato

  Per ciascuna Interaction Affordance e per l'esposizione della Thing Description si specifica uno o piú Binding Template. Sará compito dello stack software generare un endpoint per ciascun Binding Template selezionato in fase di disegno della TD. Durante la definizione delle Interaction Affordance é possibile definire il Meta-Type che quest'ultima produrrá in output (a scelta tra `application/json` oppure `text/plain`).

- **Build** (colonna di destra)
  Nella sezione di destra si permette il salvataggio e il caricamento di un file (con estensione .wotsrv) tramite i due pulsanti posizionati nella parte alta. Di seguito si richiedono le seguenti informazioni:

  - SSID e password della rete Wi-Fi a cui la Thing si dovrá connettere
  - Porte alle quali i Binding Templates risponderanno
  - Librerie Arduino esterne
  - Variabili globali e direttive *#define*
  - Codice addizionale nelle funzioni principali dello sketch (`setup()` e `loop()`)
  - Codice addizionale allo sketch

- **Terminale** (footer)
  Nel footer si trova il terminale nel quale verrá mostrato l'output del

Servient Builder. Nella parte alta si puó scegliere la tipologia di scheda prototipale su cui compilare il progetto (per ora é implementato solamente l'ESP32) ed eventualmente si puó scegliere la porta seriale della board su cui effettuare l'upload. Alla pressione del pulsante *Build & Compile* verrá realizzato il file nel linguaggio Processing contente la logica applicativa della Thing, verrá successivamente compilato e (se indicato dall'utente) effettuato il flashing sulla scheda prototipale.

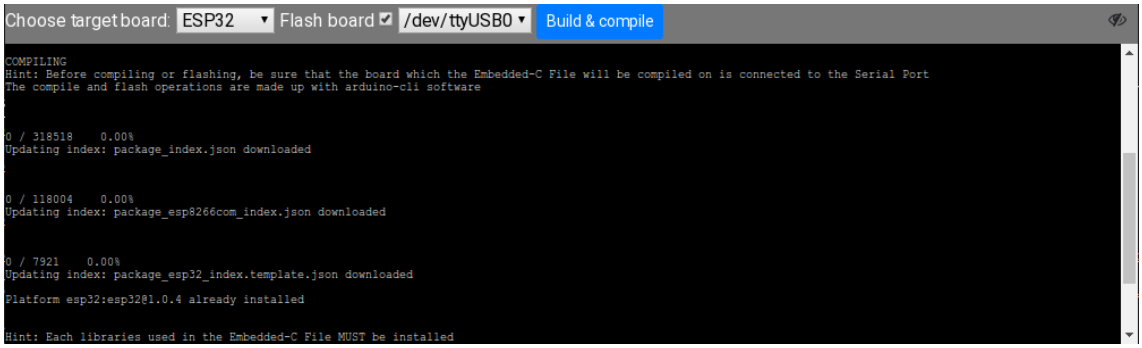

Figura 4.2: Terminale integrato nella GUI

Servient Builder (Core) Il core del progetto al suo interno contiene una CLI (interfaccia a riga di comando) per permettere la generazione del codice sorgente che dovrá poi essere compilato e caricato sul microcontrollore. Il Servient Builder puó essere utilizzato attraverso la Command Line Interface ma (come nel caso della creazione di un progetto attraverso interfaccia grafica) puó richiedere in input una Thing Description e un file di opzioni per creare lo sketch, saltando la parte di immissione tramite riga di comando. Tale core, partendo da un file di template, genera il codice Processing (compatibile per la programmazione di microcontrollori con l'interfaccia Arduino IDE) contenente gli algoritmi per implementare lo stack del WoT Servient.
Il file di template dello sketch in Processing é integrabile con altri file adatti alla generazione di codice per altri microcontrollori. Sará compito della GUI mostrare i template disponibili all'utente (e quindi la compatibilitá con una

board).

Il compito dell'interfaccia a riga di comando é mostrare il progresso della compilazione e caricamento dello sketch. Verranno riportati eventuali errori del compilatore (nel caso si sia scritto codice sintatticamente scorretto) e del flasher (che possono occorrere ad esempio se l'utente non é autorizzato ad accedere alla porta seriale del microcontrollore).

Capitolo 5

Validazione

In questo capitolo si tratterá un esempio di validazione del progetto microWoTServient, applicandolo ad un ambito reale come quello dello smart parking di biciclette. In prima istanza si tratterá il problema nel concreto dei posti occupati e disponibili in una rastrelliera e successivamente si discuterá la soluzione implementativa sviluppata.

5.1 Caso d'uso: Smart Bike Rack

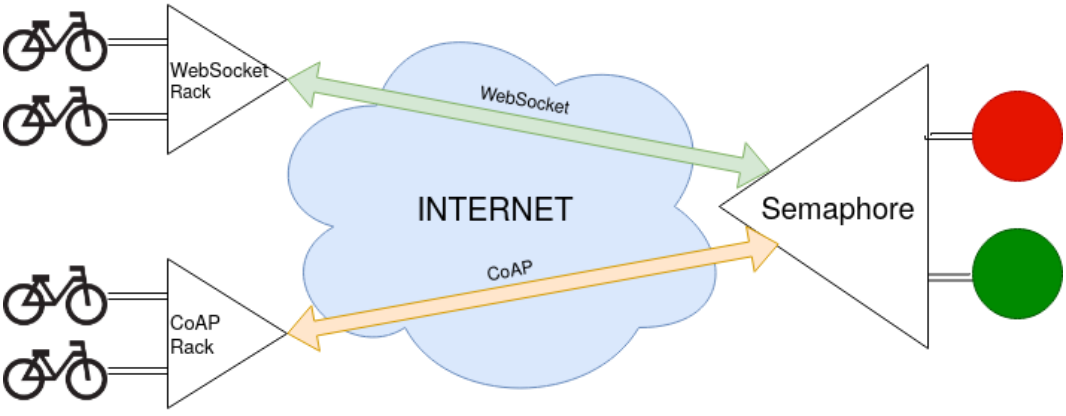

Figura 5.1: Schema Smart Bike Rack

Una *Smart Bike Rack* é una rastrelliera per biciclette che implementa l'Internet of Things per comunicare agli utenti la disponibilitá di posti disponibili per posteggiare. Tale soluzione si integra nel concetto di urbanistica intelligente (Smart City), ma applicabile anche in ambito consumer.
Gli obiettivi della Smart Bike Rack sono i seguenti:

- tenere traccia dei movimenti in entrata ed in uscita di biciclette
- monitorare l'utilizzo di servizi di bike sharing urbano
- aiutare i ciclisti a scegliere il posteggio libero piú vicino in maniera rapida
- ridurre l'utilizzo di mezzi piú inquinanti, come motociclette o autoveicoli
- mitigare la congestione del traffico cittadino

L'architettura di un sistema di Smart Bike Rack é composta dai seguenti elementi:

- **Layer di sensing**
  Rilevatori che possono essere di differente tipologia (infrarossi, prossimitá, ultrasonici, a switch, ecc...) che rilevano la presenza o meno di una bicicletta parcheggiata
- **Layer di comunicazione**
  Tecniche di intercomunicazione tra devices e di interazione con l'utente. Le tecnologie di rete devono essere caratterizzate da: affidabilitá di trasmissione, ridotto dispendio energetico, ritardo minimo.
- **Layer di orchestrazione**
  Consente di interagire con i dati rilevati dai sensori, con il fine di offrire informazioni sulla disponibilitá e sulla prenotazione di un posteggio.

- **Layer di memorizzazione**
  L'esposizione delle informazioni riguardanti lo stato dei parcheggi viene esposta da un intermediario posizionato tra le varie rastrelliere smart in una rete ed Internet attraverso API accessibili dagli utenti attraverso applicazioni mobile o web browser.

5.2 Implementazione

Per il modello di implementazione del suddetto caso d'uso si identificano due attori: il rilevatore presenza di una bicicletta e l'intermediario.
Nella fattispecie per tale esperimento si sono utilizzati tre dispositivi ESP32: due svolgono il ruolo di sensore per la rastrelliera ed uno come intermediario per unificare la comunicazione verso queste ultime.
La differenza tra i due sensori consiste nel protocollo di comunicazione per lo scambio dei dati: il primo comunica via WebSocket, il secondo via CoAP. Tale implementazione é stata testata anche con un ESP32 comunicante via HTTP. In figura 5.2 si evidenzia il modello di interazione tra le Things messe in rete.

L'intermediario implementa i vari protocolli di comunicazione utilizzati dalle Smart Rack per interfacciarsi con esse e ottenere informazioni. L'intermediario ha anche due attuatori luminori, uno rosso ed uno verde, a rappresentare il *semaforo* che mostra la disponibiltá o meno di posteggi liberi. Nonostante il modello di trasmissione e la struttura dei dati scambiati sia il medesimo tra i vari dispositivi embedded (in particolare il meta-type di trasmissione é sempre in formato JSON, e le properties disponibili sono quelle illustrate in tabella 5.2 per ciascun device), é necessario che il semaphore utilizzi entrambi i protocolli con i quali i sensori comunicano.

Nel modello di comunicazione illustrato in figura 5.2 si evidenzia il flusso dei dati. Il semaphore si sottoscrive alla ricezione di eventi da ciascun dispositivo Smart Rack. Quando tale dispositivo rileva l'ingresso oppure l'uscita

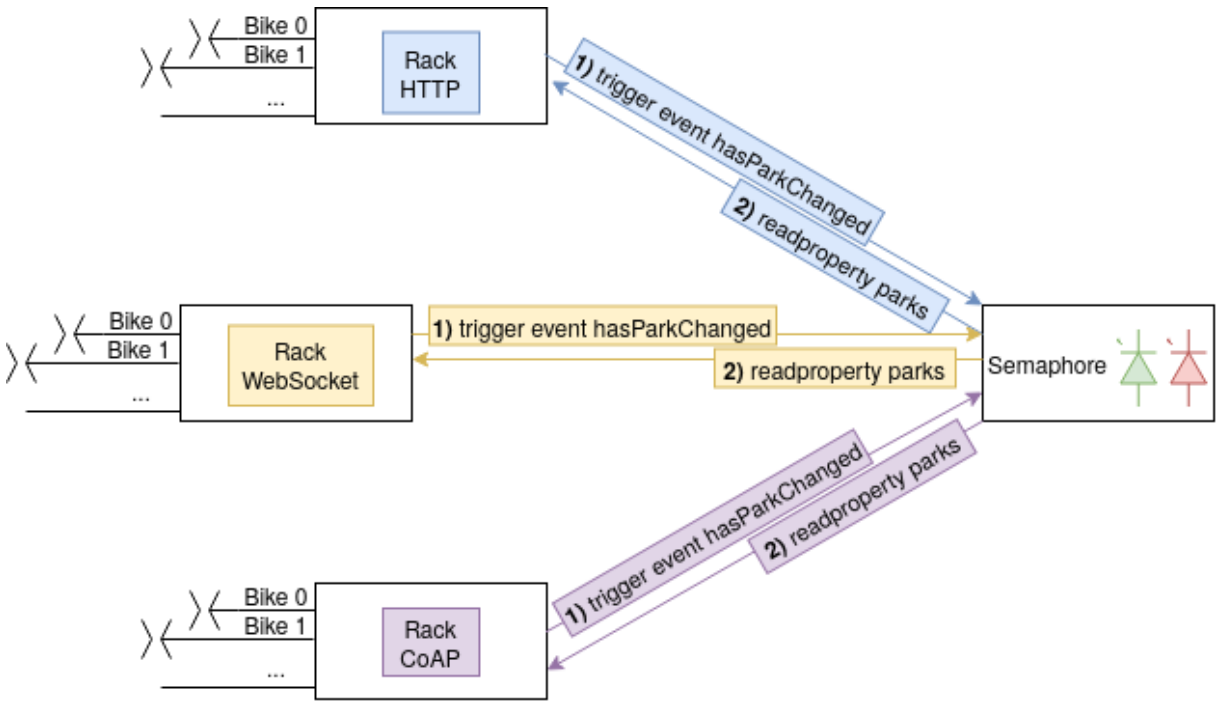

Figura 5.2: Schema di comunicazione tra le Thing "Rack" e "Semaphore"

di una bicicletta dal posteggio, viene scatenato l'evento `hasParkChanged` che verrá notificato all'intermediario. In ultima istanza il semaforo esporrá tale cambiamento attraverso il trigger di eventi, sottoscrivibili da dispositivi in rete.

Per la realizzazione di una rastrelliera prototipale sono stati utilizzati switch fisici che rilevano la presenza di un velocipede per pressione meccanica. Lo stack software é suddiviso in due sezioni: la prima parte si occupa di esporre gli eventi ai quali sottoscriversi per la notifica del cambio di stato dei posteggi (lato sensoristica), la seconda parte si occupa invece di esporre la Thing sulla rete per una vista d'insieme delle varie rastrelliere presenti in rete (lato intermediario). Il protocollo permette di ottenere, attraverso la Thing Description e le API REST esposte dal semaphore, quanti e quali posteggi sono occupati o liberi; si puó quindi progettare un interfaccia che comunichi in quale parcheggio recarsi, attraverso display monocromatici LCD oppure web

Nome	Tipo	Descrizione
`semaphore`	`String`	Puó essere ”green” o ”red”
`freeParks`	`int`	Indica il numero di parcheggi attualmente liberi
`racks`	`Array<Rack>`	Ogni elemento contiene le properties di ogni singola Rack

Tabella 5.1: Properties dell'intermediario Semaphore

Nome	Tipo	Descrizione
`name`	`String`	Rappresenta il nome della rack
`parks`	`Array<bool>`	Per ogni posteggio, true se libero, false se occupato/prenotato

Tabella 5.2: Properties della rastrelliera intelligente

browser in esecuzione su un PC pubblico, e.g. *kiosk* o *smart mirror*. La Thing Description realizzata viene illustrata dalle tabelle 5.1 e 5.2.

Entrambe le entitá (rastrelliere e semaforo) hanno in comune la Action `changeParkState` che permette la prenotazione di una rastrelliera. Se invocata sul Semaphore, questo riporterá il cambiamento sulla Rack indicata.

Una realizzazione pratica della rastrelliera smart é illustrata in figura 5.3

Figura 5.3: Applicazione reale di una Smart Bike Rack prototipale

Capitolo 6

Conclusioni

In questa tesi si é presentata la realizzazione del progetto MicroWoTServient GUI che permette la realizzazione di un WoT Servient sulla base delle indicazioni dello standard W3C WoT compatibile con microcontrollori programmabili con Arduino (ed in particolare con la board prototipale Espressif ESP32). Si é prima data una panoramica generale sull'Internet of Things e sulla problematica dell'interoperabilitá tra devices, per poi giungere alla soluzione del Web of Things come possibile protocollo mitigatore dell'eterogeneitá tra microcontrollori. Si é quindi fornita una possibile soluzione per dispositivi embedded, stilando la fase di progettazione e di implementazione dello stack software.

6.1 Stato attuale e limiti del progetto

Il lavoro svolto si é rivelato molto complesso in quanto le molte parti che lo compongono sono state strutturate e ripensate (rispetto al progetto iniziale[13]) per una futura implementazione di nuove funzionalitá. Un accento particolare é stato posto sulla possibilitá di inserire Binding Templates aggiuntivi a quelli iniziali (HTTP, WebSocket e CoAP) senza modificare profondamente il core applicativo. Uno degli obiettivi principali dell'intero

progetto é stato quello di portare un'interfaccia grafica al programma, rendendolo decisamente piú user-friendly rispetto alla sua controparte CLI.
Tra i limiti di questo progetto non si puó ignorare il fatto che lo stack del Servient aggiunge un notevole overhead al microcontrollore. In alcuni ambiti dove si richiede tutta la potenza di calcolo a dispositivi pensati per essere economici e non onerosi di risorse, questa risulta essere una criticitá.
Un'altro punto che potrebbe portare a problemi di scalabilitá riguarda il Binding Template HTTP sul device ESP32. Difatti, per un limite alla libreria ESPAsyncWebServer, si possono aprire al massimo 8 sessioni HTTP simultaneamente.

6.2 Sviluppi futuri

Il lavoro non é da considerarsi concluso, ci sono infatti molte possibilitá di ampliamento del progetto: dall'aggiunta di compatibilitá con altre schede oltre ad ESP32 (un tentativo é giá stato concluso con successo su una NodeMCU ESP8266, presente nel codice sorgente), all'identificazione di nuovi casi d'uso e all'utilizzo di protocolli di sicurezza (come HTTPS o CoAPS) con le relative criticitá legate al fatto che si sta lavorando su devices con ridotte prestazioni computazionali. Un altro punto che ritengo aperto é che la GUI, nonostante assolva a tutti i ruoli indicati nei requisiti (sottosezione 3.2) risulta essere scarna di funzionalitá accessorie, come le operazioni di undo/redo e il linting del codice inserito, rendendola cosí piú simile ad un IDE. L'intero codice é poi da tenere allineato con le indicazioni che il consorzio W3C apporta all'architettura del WoT; non é raro che vengano redatte modifiche allo standard implementato con tale lavoro.

Appendice A

Guida all'uso

In questa appendice si illusterá il funzionamento della GUI. Si illustrerá anche quali comandi sono stati messi a disposizione dal framework del MicroWoTServient.
Questa guida rappresenta lo stato dell'arte al momento della presentazione della tesi. Il progetto potrebbe subire modifiche future e/o disporre di una guida piú articolata. Per maggiori informazioni si consulti la repository del progetto su:
`https://github.com/Chello/WoT-microcontroller-servient-GUI`.

A.1 Requisiti

Il progetto funziona unicamente su sistemi Linux. Il progetto per ora supporta piena compatibilitá con la board Espressif ESP32.
Il progetto necessita delle seguenti dipendenze

- Python v3.6.x
- NodeJS v8.10.0 (testato anche con le versioni v9.11.2 e v10.21.0)
- npm v3.5.x
- pip3 package manager

- git

A.1.1 Installazione

Per l'installazione si richiede l'esecuzione di tali comandi:

```
git clone https://github.com/Chello/WoT-microcontroller-servient-GUI.git
cd WoT-microcontroller-servient-GUI
pip install .
npm install
```

A.1.2 Avvio

Per avviare il programma, una volta dentro la directory principale, si deve eseguire il comando `npm start`.

A.2 Uso

All'avvio la GUI avrá l'aspetto mostrato in figura A.1

A.2.1 Thing Description

Nella parte sinistra della GUI si costruisce la struttura della Thing Description. Nella parte superiore si trovano i pulsanti:

- **JSON**: mostra la Thing Description in formato JSON-LD che verrá prodotta. Permette anche la modifica manuale della TD
- **Properties**: permette la scelta dei campi che la Thing Description dovrá esporre. Oltre a quelli di default l'utente puó anche inserirne di nuovi manualmente

Nella sezione "Protocol Binding" si dovrá selezionare per quali Binding Template la TD dovrá essere esposta. Nelle parti sottostanti si potranno specificare Properties, Actions ed Events entrando nelle sottosezioni della TD.

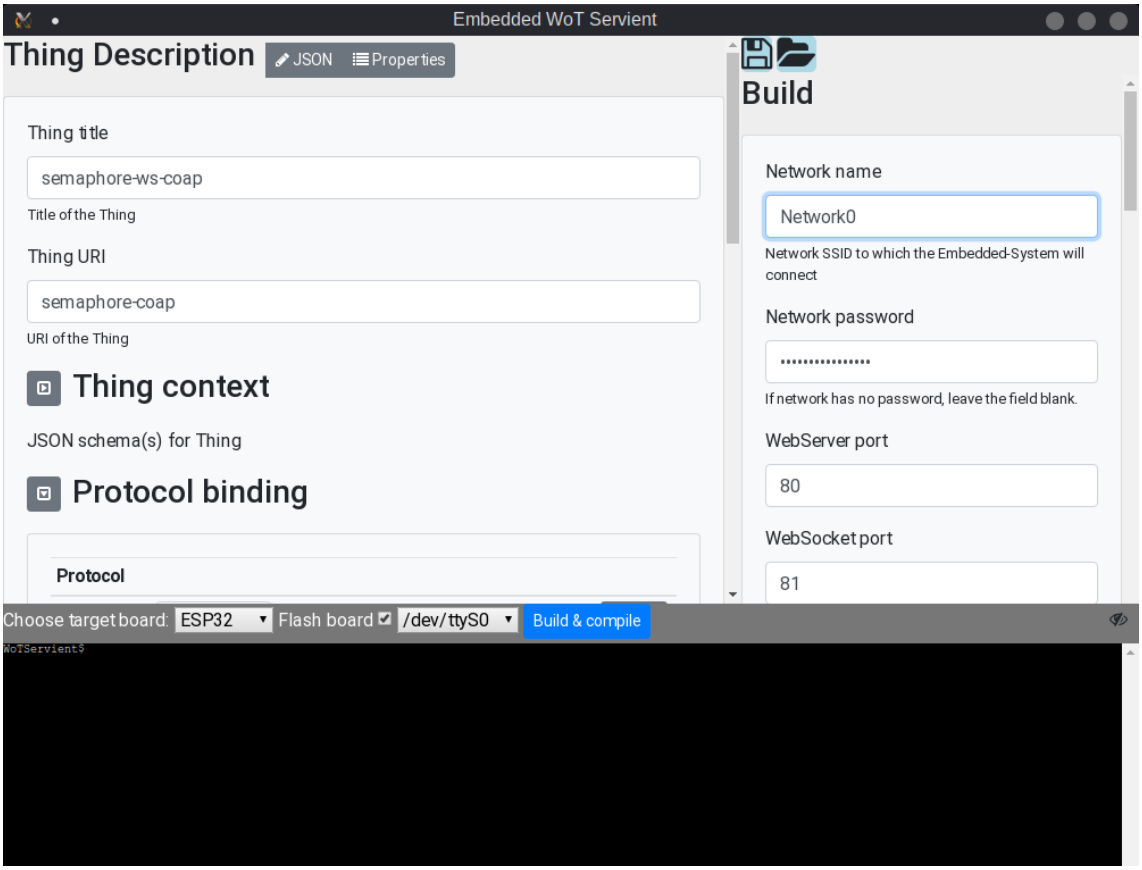

Figura A.1: Micro-WoTServient GUI

Per ogni Interaction Affordance si devono indicare a quali Binding Templates questa dovrá rispondere Per richiamare le Interaction Affordance nel codice si forniscono le seguenti indicazioni:

- **Thing properties**
  Per ogni Property che si definisce verranno generate le variabili che contengono il nome ed il valore di quest'ultima. I nomi di tali variabili sono del tipo `propertyN_name` (di tipo `String`) e `propertyN_value` (del tipo selezionato nel form), con `N` che rappresenta il numero della Property a partire da 0.

- **Thing actions**

Ogni Action definita corrisponde ad una funzione omonima, con numero e tipo dei parametri di input e tipo di output come indicato nel form. Un esempio della firma di tale funzione é:
`out_type nome_action(in_type1 param1, in_type2 param2, ...);`).
Di seguito il form richiede il corpo della funzione da invocare alla chiamata dell'Action senza includerne la firma (che verrá costruita in automatico).
N.B.: se si richiama la funzione dal codice non verranno triggerati eventuali Events collegati. Per lanciare un Event dal codice, si veda il prossimo punto.

- **Thing events**
  Per gli Events si possono definire delle Actions che, alla loro chiamata, scateneranno in automatico tale Event. In tal caso, l'output prodotto dalla funzione associata all'Action collegata verrá *parsato* come tipo `String` (se giá non lo é) e verrá utilizzato come corpo dell'evento *triggerato*.
  Per lanciare un Event dal codice si puó chiamare la funzione
  `emitEvent(String txt, String event_name)`,
  indicando il contenuto del messaggio da inviare ed il nome dell'evento da scatenare.
  Per ogni Event si deve specificare anche la condizione per la quale esso dovrá essere *triggerato*, in seguito all'invocazione di una Action collegata. Se si intende triggerare sempre l'Event alla chiamata delle Action collegate, é sufficiente riempire il campo con il valore `true`.

A.2.2 Build

Nella parte destra della GUI si definiscono le opzioni di *build* dello sketch. Verrá richiesto l'SSID e la password della rete WiFi a cui connettersi, le porte a cui i Binding Templates dovranno rispondere e librerie, macro e variabili globali addizionali. Si possono aggiungere porzioni di codice alle funzioni

`setup()` e `loop()` e definire funzioni e codice addizionale, attraverso i campi previsti dal form

A.2.3 Compile & Flash

Nella parte sottostante della GUI é presente un terminale, mostrante l'output delle fasi di *build*, *compile* e *flash* del Servient Builder. Tali fasi vengono lanciate con il pulsante "Build & Compile". Il flusso delle operazioni eseguite dal Servient Builder sono le seguenti:

- controllo della correttezza dei dati inseriti nel form Thing Description e Build (verranno mostrati gli eventuali errori)
- controllo della presenza del driver e del compilatore per la scheda selezionata
- controllo della presenza di tutte le librerie da includere (nel caso una libreria non venga trovata verrá cercata nel repository di Arduino)
- *build* dello sketch per la tipologia di scheda selezionata
- compilazione dello sketch. Se sono presenti errori sintattici il compilatore li mostrerá sul terminale
- *flash* del codice compilato sulla board. Si richiede la selezione della porta seriale a cui il microcontrollore é connesso (solamente se é selezionata la relativa *checkbox*).
- apertura del monitor seriale (se indicato dall'utente)

Bibliografia

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

[2] Kindberg T., Barton J., and Morgan J. et al. People, places, things: Web presence for the real world. Mobile Networks and Applications, 2002.

[3] S. Latre, P. Leroux, T. Coenen, B. Braem, P. Ballon, and P. Demeester. City of things: An integrated and multi-technology testbed for iot smart city experiments. In *2016 IEEE International Smart Cities Conference (ISC2)*, pages 1–8, 2016.

[4] 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.

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

[6] Romano Fantacci, Tommaso Pecorella, Roberto Viti, and Camillo Carlini. Short paper: Overcoming iot fragmentation through standard gateway architecture. In *2014 IEEE World Forum on Internet of Things (WF-IoT)*, pages 181–182. IEEE, 2014.

[7] Erik Wilde. Putting things to rest. 2007.

[8] Guinard D. D. and Trifa V. M. What is the web of things? ApacheCon Europe, Presentation, 2008.

[9] Web of things at w3c, 2020.

[10] Kovatsch M., Matsukura R., Lagally M., Kawaguchi T., Toumura K., and Kajimoto K. Web of things architecture, 2020.

[11] Kaebisch S., Kamiya T., McCool M., Charpenay V., and Kovatsch M. Web of things thing description, 2020.

[12] Reshetova E. and McCool M. Web of things security and privacy guidelines, 2020.

[13] Daniele Rossi. *Progettazione e sviluppo di una piattaforma abilitante del Web of Things per dispositivi embedded.* PhD thesis.