Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

Web of Things per dispositivi EDGE: progettazione e sviluppo di un framework per microcontrollori

Relatore:
Chiar.mo Prof.
Marco Di Felice

Correlatore:
Dott. Luca Sciullo

Presentata da:
Cristian Castiglione

Sessione III
Anno accademico 2020/2021

Abstract

La costante crescita dei dispositivi Internet of Things (IoT) ha dato luogo alla creazione e all'utilizzo di diverse tecnologie, dispositivi e relativi sistemi creando problemi di interoperabilità. Il recente standard W3C Web of Things (WoT) ha lo scopo di facilitare l'interoperabilità e l'integrazione dei dispositivi IoT con l'obiettivo di contrastare la frammentazione dell'IoT. Tuttavia, attualmente il WoT non è implementato su microcontrollori, motivo per cui in questo elaborato di tesi ci si propone di contribuire all'implementazione del paradigma WoT su tali dispositivi. Tale implementazione avviene attraverso la realizzazione di un framework multipiattaforma che permette ad utenti e sviluppatori di progettare, compilare ed installare applicazioni W3C WoT compliant su dispositivi embedded. Nello specifico, tale framework consente la creazione di Thing e Consumer creando automaticamente la Thing Description e modificando lo sketch, generato automaticamente, tramite editor e form HTML. Inoltre, fornisce supporto per i quattro più importanti protocolli di comunicazione del livello applicativo: HTTP, CoAP, WebSocket ed MQTT. L'architettura ottenuta è estendibile ad altri protocolli e ad altre funzionalità. Infine, dai risultati ottenuti si evince che questo framework non aggiunge overhead in termini di latenza nell'utilizzo degli sketch generati.

Indice

Elenco delle figure

Capitolo 1

Introduzione

Il mondo dell'Internet of Things (IoT) negli ultimi anni ha avuto una crescita esponenziale e si prevede che nel 2030 saranno 25,4 miliardi i dispositivi connessi [28]. Oltre ai vantaggi che questi dispositivi portano nel settore industriale e non solo, ci sono delle criticità già evidenti. Infatti, la crescita nell'utilizzo di questi dispositivi ha portato alla creazione di diversi protocolli e tecnologie che non consentono interoperabilità. Per questo motivo, il W3C ha proposto uno standard atto a risolvere questa criticità proponendo un paradigma basato su tecnologie esistenti e ben consolidate, le tecnologie Web [29]. Il W3C WoT ha come obiettivo quello di definire un'architettura di riferimento per le Web Things e di raggiungere l'interoperabilità tra piattaforme e domini IoT.
Nonostante il WoT sia stato progettato per essere implementato su diversi applicativi e dispositivi del panorama IoT, al momento questo paradigma vede un'implementazione ufficiale solo nel node-wot [6], un framework di alto livello che consente a sviluppatori di creare applicazioni WoT in Javascript. Per tale motivo, questo framework non è adattabile a microcontrollori e quindi su dispositivi di basso livello. Ciò rappresenta un importante ostacolo alla diffusione dello standard W3C WoT, poiché limita la possibilità di connettere al WoT un'alta percentuale di dispositivi edge IoT. Per le motivazioni appena elencate e dal momento che, come evidenziato dall'analisi di Statista

[28], i dispositivi IoT connessi crescono esponenzialmente, si è deciso di fare un lavoro grazie al quale sia possibile applicare il paradigma W3C WoT a tutti i dispositivi di basso livello.
Lo scopo di questa tesi consiste dunque nella progettazione di un framework avente come obiettivo la realizzazione di applicazioni per dispositivi IoT di basso livello applicando lo standard W3C WoT. Il framework sviluppato è multipiattaforma e consente di creare Thing e Consumer, oltre che di compilare ed installare applicazioni WoT su dispositivi embedded. Tale framework consente inoltre di modificare lo sketch generato attraverso due tipi di interfacce: editor o form HTML e di implementare i quattro protocolli di comunicazione più importanti: HTTP, CoAP, WebSocket ed MQTT.
Il risultato di questo lavoro è stato raggiunto attraverso uno studio approfondito dello stato dell'arte del panorama IoT, attraverso un'analisi dei bisogni e dei limiti attualmente presenti in questo contesto e attraverso lo studio di una architettura flessibile in grado di cambiare nel tempo e di adattarsi a contesti IoT.
Attraverso questo elaborato di tesi si contribuisce a portare il paradigma W3C WoT sui microcontrollori attraverso un framework multipiattaforma che consente ad utenti e sviluppatori di generare sketch W3C WoT compliant in grado di essere eseguiti su dispositivi IoT con durata energetica e risorse computazionali limitate e non aggiungendo overhead ai protocolli disponibili. La possibilità che offre questo framework per l'utilizzo di soluzioni WoT per sistemi embedded migliorerebbe definitivamente il processo di sviluppo di applicazioni W3C WoT e l'applicazione di questo paradigma su tutti i dispositivi di basso livello.
Questo elaborato di tesi è strutturato nel seguente modo: nel secondo capitolo viene descritto lo stato dell'arte dell'Internet of Things, del Web of Things e dello standard W3C Web of Things. Nel terzo capitolo si entra nel dettaglio del progetto descrivendo gli obiettivi, i requisiti e l'architettura del framework. Nel quarto capitolo vengono descritte le tecnologie utilizzate per la realizzazione del framework e i vari componenti realizzati. Nel quinto

capitolo viene trattata la parte di validazione, quindi viene effettuata la valutazione delle performance attraverso la realizzazione di scenari reali. Infine, nel sesto capitolo vengono esposte le conclusioni e gli sviluppi futuri.

Capitolo 2

Stato dell'arte

2.1 Internet of things

Il termine Internet of Things (IoT) descrive diverse tecnologie e discipline di ricerca che consentono a Internet di raggiungere il mondo reale degli oggetti fisici. Il termine Internet of Things fu coniato nel 1999 dal ricercatore Kevin Ashton [4]. L'idea principale alla base dell'IoT è quella di avere una connessione indipendente e autonoma, sicura e che consenta lo scambio di dati tra dispositivi fisici del mondo reale e applicazioni reali [16]. L'IoT trasforma gli oggetti tradizionali in oggetti intelligenti sfruttando tecnologie di comunicazione, sensori, protocolli di comunicazione e applicazioni internet. Nel corso degli ultimi anni l'Internet of Things ha trovato applicazione in ambienti domestici e non solo aziendali, questo ha portato l'IoT ad avere una crescita esponenziale nell'arco di una decina di anni [8]. Si prevede che il numero di dispositivi IoT nel mondo quasi triplicherà da 8,74 miliardi nel 2020 a oltre 25,4 miliardi di dispositivi IoT nel 2030 [28]. Nel 2020, il maggior numero di dispositivi IoT si trova in Cina con 3,17 miliardi di dispositivi. I dispositivi IoT sono utilizzati in tutti i tipi di settori verticali e mercati di consumo, con il segmento dei consumatori che rappresenta circa il 60% di tutti i dispositivi IoT connessi IoT. Si prevede che questa quota rimarrà a questo livello nei prossimi dieci anni [28]. In figura 2.1 è possibile vedere il

grafico con la previsione.

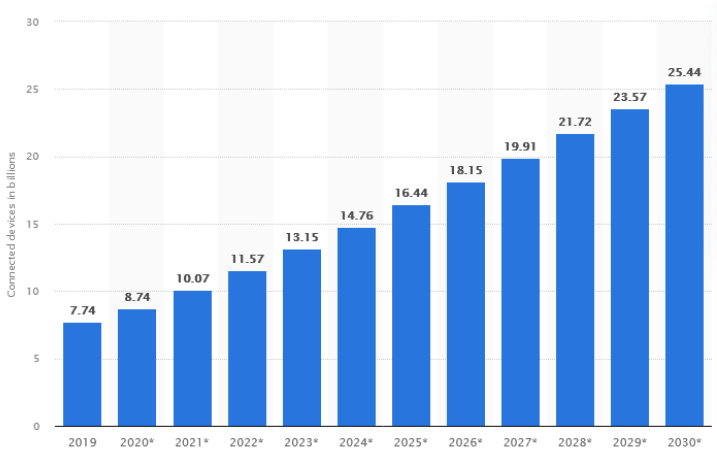


Figura 2.1: Crescita dei dispositivi IoT [8]

Figura 2.2: Principali categorie di applicazione IoT [27]

L'IoT oggi gioca un ruolo fondamentale in diversi settori, come ad esempio nei trasporti, assistenza sanitaria, automazione industriale, in ambienti

domestici, nell'agricoltura ecc [8]. In figura 2.2 è possibile visualizzare uno scenario generico dell'IoT.
Infatti, dopo il World Wide Web e la tecnologia mobile, l'IoT rappresenta la rivoluzione tecnologica più travolgente della nostra vita in quanto stiamo vivendo un cambio di paradigma in cui gli oggetti di uso quotidiano diventano interconnessi e intelligenti [28]. Questi dispositivi sono generalmente dotati di diversi tipi di sensori e attuatori in grado di connettersi e comunicare su Internet. Il flusso di lavoro, semplificato, di base dell'IoT può essere descritto come segue:

- Rilevamento degli oggetti, identificazione e comunicazione di informazioni specifiche sull'oggetto. Le informazioni sono i dati rilevati su temperatura, orientamento, movimento, vibrazioni, accelerazione, umidità, cambiamenti chimici nell'aria ecc. a seconda del tipo di sensori. Una combinazione di diversi sensori può essere utilizzata per la progettazione di servizi intelligenti;

- Organizzare un'azione. Le informazioni sull'oggetto ricevute vengono elaborate da un dispositivo/sistema intelligente che determina quindi un'azione automatizzata da invocare;

- Il dispositivo/sistema intelligente fornisce servizi avanzati e include un meccanismo per fornire feedback all'amministratore sullo stato corrente del sistema e sui risultati delle azioni richiamate [16].

Il collegamento a Internet di un gran numero di oggetti IoT dotati di sensori, genera quelli che vengono chiamati "big data". I dispositivi IoT necessitano di meccanismi per archiviare, elaborare e recuperare i dati, in quanto generano un gran numero di dati al secondo che a loro volta richiedono calcoli complessi per estrarre conoscenza. Pertanto, le risorse di storage e di elaborazione del cloud rappresentano la scelta migliore per l'IoT per archiviare ed elaborare big data [8].

L'IoT offre enormi vantaggi nei settori in cui può essere applicato ma ci sono ancora molte sfide da affrontare, alcune di essi sono riportate di seguito:

1. Naming e Identity Management: l'IoT collegherà miliardi di oggetti per fornire servizi innovativi. Ogni oggetto/sensore deve avere un'identità univoca su Internet. Pertanto, è necessario un efficiente sistema di denominazione e gestione dell'identità in grado di assegnare e gestire dinamicamente l'identità univoca per un numero così elevato di oggetti;

2. Interoperability and Standardization: molti produttori forniscono dispositivi utilizzando le proprie tecnologie e servizi che potrebbero non essere accessibili da altri. La standardizzazione dell'IoT è molto importante per fornire una migliore interoperabilità per tutti gli oggetti e i dispositivi sensori;

3. Information Privacy: l'IoT utilizza diversi tipi di tecnologie di identificazione degli oggetti, ad esempio RFID, codici a barre 2D, ecc. Poiché ogni tipo di oggetto di uso quotidiano porterà questi tag di identificazione e incorporerà le informazioni specifiche dell'oggetto, è necessario adottare adeguate misure di privacy e prevenire l'accesso non autorizzato;

4. Objects safety and security: l'IoT è costituito da un numero molto elevato di oggetti di percezione che si estendono su una certa area geografica, è necessario impedire l'accesso malevolo agli oggetti che possono causare danni fisici a loro o possono modificarne il funzionamento;

5. Data confidentiality and encryption: i sensori eseguono rilevamenti o misurazioni indipendenti e trasferiscono i dati all'unità di elaborazione delle informazioni tramite il sistema di trasmissione. È necessario che i sensori dispongano di un meccanismo di crittografia adeguato per garantire l'integrità dei dati nell'unità di elaborazione delle informazioni. Il servizio IoT determina chi può vedere i dati, quindi è necessario proteggere i dati dagli esterni;

6. Network security: i dati dai dispositivi con sensori vengono inviati su una rete di trasmissione cablata o wireless. Il sistema di trasmissione

dovrebbe essere in grado di gestire i dati provenienti da un gran numero di dispositivi con sensori senza causare alcuna perdita di dati a causa della congestione della rete, garantire adeguate misure di sicurezza per i dati trasmessi e prevenirli da interferenze o monitoraggio da individui esterni;

7. Spectrum: i dispositivi con sensori richiederanno uno spettro dedicato per trasmettere i dati sul supporto wireless. A causa della limitata disponibilità dello spettro, è necessario un efficiente meccanismo di allocazione dello spettro dinamico per consentire a miliardi di sensori di comunicare attraverso la rete senza fili;

8. Greening of IoT: il consumo di energia della rete sta aumentando a un ritmo molto elevato a causa dell'aumento della velocità dei dati, dell'aumento del numero di servizi abilitati a Internet e della rapida crescita dei dispositivi edge connessi a Internet. Il futuro IoT causerà un aumento significativo del consumo di energia della rete. Pertanto, è necessario adottare tecnologie "green" per rendere i dispositivi di rete il più efficienti possibile dal punto di vista energetico.

2.1.1 Stack architetturale

L'IoT è in grado di interconnettere miliardi di dispositivi attraverso Internet, quindi c'è bisogno di un'architettura a strati flessibile. Il sempre crescente numero di architetture proposte non è ancora convergente ad a modello di riferimento. Nel frattempo, ci sono alcuni progetti che cercano di progettare un'architettura comune basata su l'analisi dei bisogni dei ricercatori e dell'industria. Il modello base è l'architettura a 3 livelli [8]. I tre livelli dell'architettura sono:

Perception layer

in questo livello sono presenti i sensori che raccolgono le informazioni. Questo livello rappresenta i sensori fisici dell'IoT che mirano a raccogliere ed elaborare le informazioni. Questo livello include sensori e

attuatori per eseguire diverse funzionalità come interrogare posizione, temperatura, peso, movimento, vibrazioni, accelerazione, umidità, ecc. Il livello di percezione digitalizza e trasferisce i dati al livello di astrazione dell'oggetto attraverso canali sicuri. I big data creati dall'IoT vengono avviati a questo livello.

Network layer

fornisce il supporto per la trasmissione dei dati, si basa sulla rete e sulla comunicazione per facilitare il processo di transazione. I dati possono essere trasferiti attraverso varie tecnologie come RFID, 3G, GSM, UMTS, WiFi, Bluetooth LowEnergy, infrarossi, ZigBee, ecc. Inoltre, altre funzioni come il cloud computing e i processi di gestione dei dati sono gestiti a questo livello.

Application layer

presenta un'interfaccia utente per l'interazione con i dispositivi. Il livello dell'applicazione fornisce i servizi richiesti dai clienti. Ad esempio, il livello dell'applicazione può fornire misurazioni della temperatura e dell'umidità dell'aria al cliente che richiede tali dati. L'importanza di questo livello per l'IoT è che ha la capacità di fornire servizi intelligenti di alta qualità per soddisfare le esigenze dei clienti. Il livello di applicazione copre numerosi mercati verticali come la casa intelligente, l'edilizia intelligente, i trasporti, l'automazione industriale e l'assistenza sanitaria intelligente [8] [11].

Recentemente sono stati proposti altri modelli che aggiungono più astrazione all'architettura IoT portando i layer da 3 a 5 [8], per affrontare molti fattori come scalabilità, interoperabilità, affidabilità, QoS, ecc. Nella figura 2.3 è possibile vedere differenti tipologie di architetture.

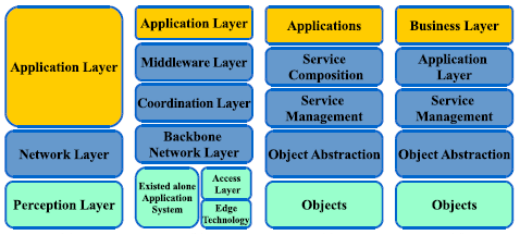


Figura 2.3: Principali modelli di architettura IoT [8]

Gli altri due layer aggiunti sono i seguenti:

Service Management Layer

Il Service Management Layer o Middleware che accoppia un servizio con il suo richiedente in base a indirizzi e nomi. Questo livello consente ai programmatori di applicazioni IoT di lavorare con oggetti eterogenei senza considerare una piattaforma hardware specifica. Inoltre, questo livello elabora i dati ricevuti, prende decisioni e fornisce i servizi richiesti tramite i protocolli di rete.

Business Layer

Il livello business gestisce le attività dei servizi complessivi del sistema IoT. Le responsabilità di questo livello sono di costruire un modello di business, grafici, diagrammi di flusso, ecc. basati sui dati ricevuti dal livello Applicazione. Dovrebbe anche progettare, analizzare, implementare, valutare, monitorare e sviluppare elementi relativi al sistema IoT. Il livello aziendale lo rende possibile per supportare i processi decisionali basati sull'analisi dei Big Data [8].

2.1.2 Tecnologie di comunicazione

Per far sì che gli oggetti fisici comunichino le informazioni acquisite, questi devono essere dotati di tecnologie in grado di avviare una comunicazione in cui instradare i dati. Le tecnologie utilizzate in questo ambito devono

sfidarsi con le basse prestazioni e capacità dei dispositivi IoT, per cui lo scenario ideale di comunicazione avviene attraverso tecnologie wireless a bassa potenza.

Attualmente esistono diverse tecnologie e protocolli di comunicazione che è possibile sfruttare nei dispositivi IoT in base allo scopo del progetto e alle capacità del dispositivo. Le tecnologie di comunicazione nel mondo IoT vengono suddivise in due categorie:

Tecnologie di comunicazione a corto raggio

Fin dall'inizio della sua definizione, il concetto di IoT è stato principalmente associato alle tecnologie di comunicazione di prossimità. Tra queste tecnologie rientrano ad esempio l'identificazione a radiofrequenza (RFID), che è stata la tecnologia pioniera relativa all'IoT e la prima realtà a collegare gli oggetti fisici nel mondo reale con la loro rappresentazione attraverso le informazioni [22]; oppure metodi di Automatic Identification (Auto-ID) come Barcode, QR-code e Near Field Communication (NFC) [5]. Tra le tecnologie a corto raggio più importanti troviamo anche la tecnologia Bluetooth, ZigBee, IPv6 Over Low Power WPANs (6LoWPAN) e Z-Wave [22].

Tecnologie di comunicazione a lungo raggio (LPWAN)

Oltre alle tecnologie di comunicazione a corto raggio, nel contesto IoT esistono diverse tecnologie a lungo raggio, tra queste troviamo: Weightless-N, Weightless-P, Weightless-W, Ingenu MN, SigFox e LoRa [22].

2.2 Web of things

Il web ha permesso a Internet di evolversi da una rete di computer che scambiano bit di dati a una piattaforma di servizi mondiale accessibile attraverso protocolli standard e universalmente conosciuti come HTTP. Allo stesso modo, l'Internet of Things ha bisogno del proprio livello applicativo

per essere accessibile da tutti. Proprio come Internet aveva bisogno del Web, l'IoT ha bisogno di una serie di standard che le applicazioni possono utilizzare per controllare, monitorare e aggregare i dati delle Things connesse [10].

Nel campo dell'informatica pervasiva c'è molta ricerca sull'integrazione di oggetti fisici con il mondo digitale. I recenti sviluppi nel campo dei dispositivi embedded hanno portato le Things a popolare sempre più la nostra vita quotidiana, formando lentamente ma costantemente reti interconnesse di oggetti fisici. Come già accennato nella sezione precendente, per facilitare queste connessioni, la ricerca e l'industria hanno messo a punto una serie di protocolli di rete e di trasporto a bassa potenza come Zigbee, Bluetooth, IEEE 802.15.4 o, più recentemente, WiFi a bassa potenza e 6LoWPAN. Tuttavia, mentre questi sviluppi aiutano a integrare le Things a livello di rete, a livello di applicazione i dispositivi embedded sono ancora isolati e incompatibili. Di conseguenza, le Things rimangono difficili da integrare in diverse applicazioni. Diverse piattaforme di servizio propongono un'architettura integrata standardizzata per facilitare l'integrazione tra Things. Standard come UPnP e DNLA, Jini o OSGi affrontano con successo problemi come il rilevamento e la registrazione del servizio. Tuttavia, questi sistemi non sono completamente compatibili tra loro e la loro complessità e la mancanza di strumenti ben noti consentono loro di raggiungere solo una comunità relativamente piccola di sviluppatori esperti. Pertanto, il loro utilizzo diretto è stato finora piuttosto limitato. Nel mondo dell'informatica aziendale, l'interoperabilità e l'accoppiamento libero si ottengono utilizzando i servizi Web WS-*. I servizi Web WS-*, si basano su due principali formalismi XML: WSDL e SOAP, nonché su un insieme di standard aggiuntivi (WS-Addressing, WS-Security, WSDiscovery, ecc.), l'approccio WS-* è un miglioramento rispetto ai protocolli proprietari tradizionalmente utilizzati in questo campo, tuttavia presenta anche importanti carenze. Gli standard WS-* sono piuttosto dettagliati e pesanti in termini di larghezza di banda, memoria e CPU richieste. Questo li rende difficili da implementare su dispositivi con risorse limitate e ancora più importante, i servizi Web utilizzano effettivamente il Web come livello

di trasporto e non come architettura dell'applicazione rendendoli più difficili da utilizzare e integrare con il World Wide Web. Oltre a Internet, il Web illustra bene come un insieme di standard relativamente semplici e aperti (ad es. HTTP, HTML, XML, JSON, ecc.) possano essere utilizzati per costruire sistemi molto flessibili preservando l'efficienza e la scalabilità. L'integrazione e gli sviluppi di applicazioni sul Web, insieme alla sua disponibilità onnipresente su un'ampia gamma di dispositivi (ad esempio, computer desktop, laptop, telefoni cellulari, dispositivi di gioco, ecc.) rendono il Web un candidato eccezionale per una piattaforma di integrazione universale. Pertanto, poiché sempre più dispositivi si connettono a Internet, è possibile utilizzare il World Wide Web e le relative tecnologie come piattaforma per le Things. Il Web of Things (WoT), può essere considerato come un perfezionamento dell'Internet delle Things integrando le Things non solo su Internet (la rete), ma nel Web (il livello dell'applicazione). Per ottenere questa integrazione vengono implementati server Web sulle Things e viene utilizzata l'architettura REST (Representational State Transfer). Il concetto di base dell'architettura REST è che tutto è modellato come "risorsa", o in particolare risorse HTTP, con un Universal Resource Identifier (URI). L'architettura REST si basa sui seguenti quattro principi:

1. Identificazione delle risorse tramite URI. Tutte le risorse esposte dai servizi web RESTful sono identificate da URI. Attraverso URI, i client possono identificare i loro obiettivi di interazione. Viene fornito uno spazio di indirizzamento globale per la scoperta di servizi e risorse;

2. Interfaccia uniforme. I servizi RESTful trattano HTTP come un protocollo applicazione anziché un protocollo di trasporto come in WS-*. Pertanto, il termine REST viene spesso utilizzato insieme ad HTTP e le risorse RESTful possono essere manipolate utilizzando verbi HTTP come PUT, GET, POST e DELETE. PUT crea una nuova risorsa mentre DELETE la elimina. GET recupera lo stato corrente di una risorsa in alcune rappresentazioni mentre POST aggiorna una risorsa con un nuovo stato;

3. Messaggi autodescrittivi. Le risorse sono disaccoppiate dalle loro rappresentazioni in modo tale che ci sia libertà nell'utilizzare una varietà di formati di dati per descrivere se stesse, a condizione che i formati di rappresentazione appropriati siano concordati e comprensibili dagli endpoint. Ad esempio, i dati possono essere in qualsiasi formato di uso comune come HTML, XML, testo normale, PDF e JPEG. I metadati relativi alla risorsa possono essere utilizzati per controllare la memorizzazione nella cache, rilevare errori di trasmissione, negoziare il formato di rappresentazione ed eseguire l'autenticazione o il controllo dell'accesso tra gli endpoint;

4. Operazioni stateless. Ogni interazione con una risorsa stessa è stateless. Tuttavia, le interazioni stateless possono essere realizzate tramite collegamenti ipertestuali. Lo stato di una risorsa può essere trasferito in modo esplicito mediante riscrittura URI, cookie e campi modulo nascosti. Gli stati possono anche essere incorporati in un messaggio di risposta per interazioni stateless [32].

L'architettura RESTful è preferita per il WoT principalmente per le seguenti funzionalità. Uno è la sua bassa complessità e l'altro sono le sue interazioni stateless e il loose-coupling. Le due funzionalità consentono ai server Web nell'architettura RESTful di essere incorporati in dispositivi con risorse limitate e consentono anche una facile composizione di servizi Web. Con la disponibilità dei "tiny" Web Server, cioè degli http server basilari, le Things possono essere astratte come servizi Web e integrate perfettamente nel Web esistente. Utilizzando tecnologie esistenti del Web è possibile unificare il mondo cibernetico con il mondo fisico, in questo senso è possibile considerare l'IoT come Web of Things (WoT). Grazie al fatto che le tecnologie Web esistenti possono essere riutilizzate e adattate per creare nuove applicazioni e servizi IoT. A differenza della visione tradizionale dell'IoT che attribuisce un indirizzo IP ai dispositivi di uso quotidiano e li rende interconnessi su Internet, WoT consente loro di parlare la stessa lingua, in modo da comunicare e interagire liberamente sul Web, arricchendo così la portata dei servizi web

tradizionali aggiungendo servizi del mondo fisico [32]. L'interazione con le Things può anche avvenire quasi interamente in un Browser, uno strumento che è sempre disponibile e con cui la maggior parte degli utenti ha familiarità. È possibile creare applicazioni su di loro utilizzando linguaggi e tecnologie ben note del Web. Inoltre, le Things possono trarre vantaggio dai meccanismi che hanno reso il Web scalabile e di successo come la memorizzazione di informazioni nella cache, il load-balancing, l'indicizzazione e la ricerca [9]. Ne risulta che Wil eb of Things è un approccio in cui le risorse sono disponibili attraverso meccanismi Web standard. Ad esempio, in una rete di sensori, ogni sensore ha un URI (è una risorsa) e può essere interrogata per le sue letture (ha uno stato che può essere recuperato accedendo alla risorsa). Il grande vantaggio di questo approccio di integrazione è che le Things possono quindi essere trattate come qualsiasi altra risorsa Web, il che significa che possono essere riutilizzate in contesti e applicazioni diversi e possono essere utilizzate in un modo molto più aperto rispetto alla semplice esposizione tramite API che spesso limitano l'accesso e le interazioni a un insieme molto specifico di potenziali utenti. L'approccio dell'integrazione completa delle Things nel Web è un passo verso la visione del Physical Computing, in cui l'interazione con le Things si fonde perfettamente con il Web esistente [30].

Il concetto di Web of Things esplora come è possibile riutilizzare le tecnologie del Web per far dialogare dispositivi embedded e far transitare i loro dati in contesti dove possono essere sfruttati per costruire applicazioni innovative. L'architettura del Web of Things si basa su 4 livelli: Access, Find, Share & Secure e Compose, come è possibile vedere in figura 2.4. Ogni livello risolve una serie di problemi utilizzando le tecnologie Web per il livello sopra di esso. Ad esempio, il livello Access riguarda la creazione di API Web per le Things, mentre il livello Find presuppone che queste API esistano e si occupa di renderle rilevabili e trovabili sul Web [10].

2.2.1 Architettura del Web of Things

L'obiettivo generale di questa architettura è facilitare l'integrazione delle Things con i servizi esistenti sul Web e facilitare la creazione di applicazioni Web utilizzando le Things. In particolare, l'architettura proposta soddisfa quattro requisiti generali:

- Facile accesso agli sviluppatori e avere una rapida prototipazione. Ciò consente di aumentare la platea di sviluppatori, persone con competenze tecnologiche o ricercatori di sviluppare Things e contribuire a promuovere l'innovazione (pubblica) utilizzando Things;
- Offrire accesso diretto agli utenti. Ciò consente agli utenti di accedere facilmente e utilizzare le Things senza la necessità di installare software aggiuntivo. Da un browser Web (o da una libreria HTTP nel caso di un client software) hanno la possibilità di estrarre, salvare e condividere direttamente dati e servizi di oggetti intelligenti. Ciò garantisce l'usabilità dell'architettura e riduce al minimo le barriere d'ingresso per gli utenti;
- Offrire un accesso "lightweight" ai dati delle Things. Ciò consente di creare applicazioni in cui i dati del mondo reale vengono consumati direttamente da dispositivi con risorse limitate come telefoni cellulari o nodi di sensori wireless senza richiedere software dedicato su questi dispositivi.

A differenza delle tradizionali architetture a strati come il modello OSI (Open Systems Interconnection), i livelli nell'architettura Web of Things, mostrati in figura 2.4, presentata non sono rigorosamente definiti e non nascondono letteralmente i livelli precedenti. Piuttosto, l'architettura dovrebbe essere vista come un ecosistema di servizi diversi che facilitano, passo dopo passo, la creazione di applicazioni utilizzando oggetti intelligenti. Tuttavia, le applicazioni possono essere costruite sopra i servizi offerti dall'implementazione di ogni livello o sopra una combinazione di essi, a seconda dei requisiti di un particolare caso d'uso.

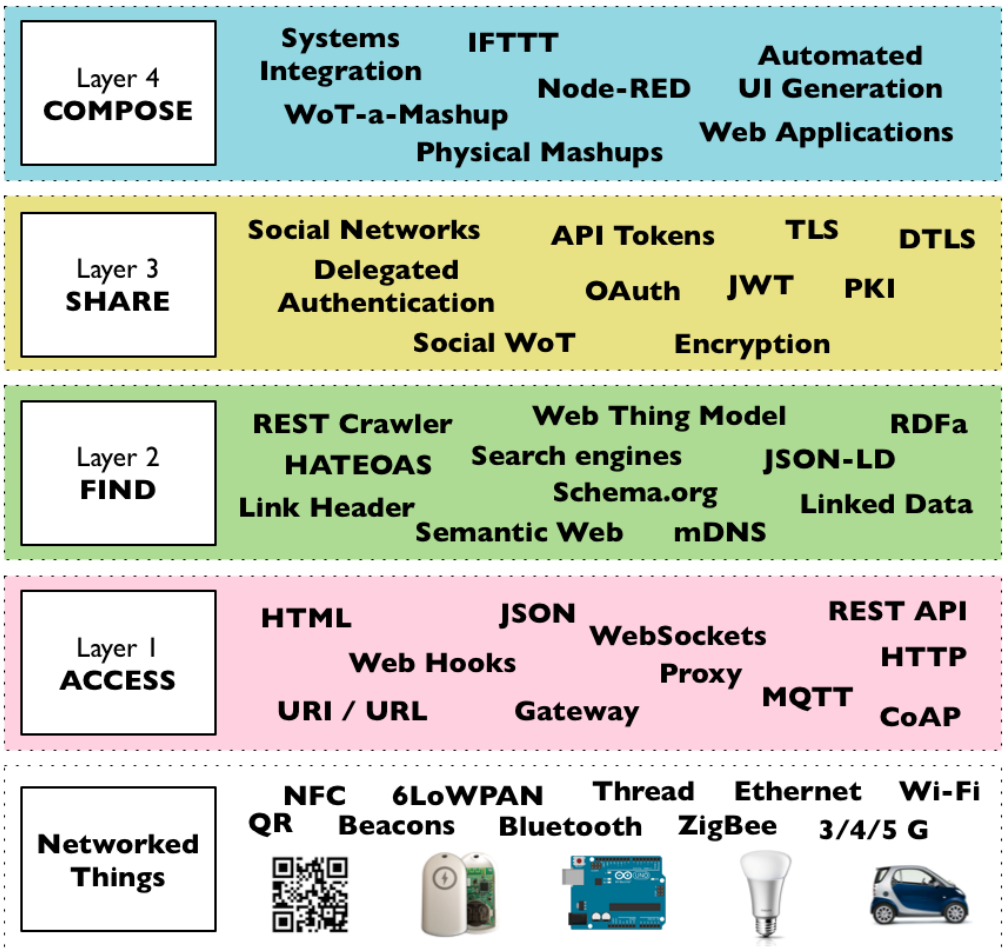


Figura 2.4: Architettura WoT a 4 layer [10]

2.2.2 Accessibility layer

Questo livello si occupa della trasformazione delle Things in Web Things in modo da integrare facilmente le Things nel Web esponendo i propri servizi tramite API RESTful utilizzando HTTP, basato su TCP/IP e sul formato dati JSON. L'access layer descrive anche come usare WebSocket per adattarsi al fatto che un certo numero di casi d'uso IoT forniscono dati in tempo reale o dati basati su eventi [9] [10].

2.2.3 Findability layer

Con l'access layer si garantisce che le Things siano accessibili sul Web, quindi diventano parte integrante del Web. Sebbene ciò presenti numerosi vantaggi, solleva anche importanti sfide. Tra questi c'è la ricerca e la ricerca di servizi rilevanti, quindi questo livello si occupa di rendere le Things trovabili. Una tecnica è renderli ricercabili, proprio come una pagina Web viene indicizzata attraverso i motori di ricerca anche una Thing può trarre vantaggio dall'essere indicizzata sui motori di ricerca. Nel libro "A Web of Things Application Architecture - Integrating the Real-World into the Web" viene proposto l'uso di un modello descrittivo implementato attraverso annotazioni semantiche. Essere accessibili digitalmente non è sinonimo di essere comprensibili dalla macchina. Ad esempio, la copertina di una pubblicazione può contenere una foto accessibile digitalmente, ma il significato di quella foto non è comprensibile dalla macchina. Ma un codice a barre su quella copertina è consumabile dalla macchina in quanto consente a un programma di identificare l'oggetto e probabilmente di accedere al suo costo e tracciarne l'acquisto e quindi consente di accedere a tutte le sue informazioni. L'utilizzo di RDFa (Resource Description Framework in Attributes) su una pagina Web ha uno scopo equivalente al codice a barre. Consente a un motore di ricerca di identificare il significato dei dati digitali, rendendoli dati strutturati. RDF fornisce un meccanismo per esprimere dati e relazioni. RDF in Attributes (RDFa) è un linguaggio che consente di esprimere i dati RDF all'interno di un documento HTML. Ciò consente alla Thing di essere leggibile sia dalla macchina che dall'uomo [9] [10].

2.2.4 Sharing layer

Con l'accessibility layer dell'architettura Web of Things si garantisce che le Things vengano integrate nel Web. Il findability layer consente agli esseri umani e alle applicazioni di trovare i servizi delle Things. Lo sharing layer si occupa della condivisione dei servizi offerti dalle Thing, consentendo l'accesso

ai servizi come risorse Web in maniera sicura [9] [10].

2.2.5 Composition layer

Una volta che le Things sono sul Web (access layer) dove possono essere trovate da esseri umani e macchine (find layer) e le loro risorse possono essere condivise in modo sicuro con gli altri (share layer), bisogna considerare l'integrazione di dati e servizi delle Things in un ecosistema di strumenti Web come software di analisi e piattaforme mashup. L'obiettivo del livello compose è rendere ancora più semplice la creazione di applicazioni che coinvolgono oggetti e servizi Web. Gli strumenti del livello compose vanno da toolkit Web (ad esempio, SDK JavaScript che offrono astrazioni di livello superiore) a dashboard con widget programmabili e infine a strumenti di mashup fisici come Node-RED. Ispirati dai servizi partecipativi Web 2.0 e in particolare dai Web mashup, i mashup fisici offrono una visione unificata del Web classico e del Web of Things e consentono alle persone di creare applicazioni utilizzando i servizi WoT senza richiedere competenze di programmazione. Un altro aspetto del livello compose è guardare il significato dei dati in quanto il lato più interessante dell'IoT è l'enorme quantità di dati che genera [9] [10].

2.3 W3C Web of things

Il World Wide Web Consortium (W3C), è un'organizzazione non governativa internazionale che ha come scopo quello di favorire lo sviluppo di tutte le potenzialità del World Wide Web e diffondere la cultura dell'accessibilità della Rete. La principale attività svolta dal W3C consiste nello stabilire standard tecnici per il World Wide Web inerenti sia i linguaggi di markup che i protocolli di comunicazione [31]. Nel WoT il W3C ha realizzato uno standard per facilitare l'interoperabilità e l'integrazione dei dispositivi IoT con l'obiettivo di contrastare la frammentazione dell'IoT e la sua tendenza a formare applicazioni verticali [17]. In generale, l'architettura W3C WoT è progettata per descrivere ciò che esiste piuttosto che per descrivere cosa

implementare [29]. Nei progetti IoT, gli sviluppatori devono affrontare situazioni impegnative, devono comprendere un panorama tecnologico eterogeneo costituito da diversi sistemi e servizi IoT di diversi fornitori e produttori. Questa diversità include variazioni nei protocolli di comunicazione, modelli di dati per lo scambio di dati e requisiti di sicurezza. Le applicazioni IoT vengono generalmente sviluppate utilizzando uno sforzo elevato applicato a un caso d'uso ristretto e specifico. Durante la loro vita, tali applicazioni sono difficili da estendere, mantenere o riutilizzare. Il Web of Things (WoT) fornisce una serie di blocchi tecnologici standardizzati che aiutano a semplificare lo sviluppo di applicazioni IoT seguendo il paradigma Web già ben consolidato. Questo approccio aumenta la flessibilità e l'interoperabilità, in particolare per le applicazioni tra domini, oltre a consentire il riutilizzo di standard e strumenti consolidati. Il W3C WoT fornisce documenti normativi ed informativi, i documenti normativi definiscono le specifiche che devono essere soddisfatte per applicare lo standard proposto, mentre i documenti informativi sono di supporto ai documenti normativi [29].

2.3.1 Casi d'uso

In questa sezione vengono presentati i domini applicativi e i casi d'uso presi in considerazione dal W3C WoT che vengono utilizzati per derivare l'architettura astratta che verrà discussa nella sezione 2.3.12 [29].

Settore consumer

Nel settore consumer, che ha come caso d'uso la Smart Home, ci sono molteplici risorse che traggono vantaggio dall'essere connessi, come ad esempio luci e condizionatori che possono essere spenti in base all'occupazione della camera, le tapparelle possono essere chiuse automaticamente in base alle condizioni meteorologiche, il consumo di energia e di altre risorse può essere ottimizzato in base a modelli di utilizzo o basandosi sulle previsioni. Un esempio di Smart Home è mostrato in figura 2.5. In questo caso,

i gateway sono collegati a dispositivi come sensori, telecamere ed elettrodomestici attraverso corrispondenti protocolli di comunicazione locale come KNX, ECHONET, ZigBee, DECT ULE e Wi-SUN. I gateway possono essere collegati al cloud tramite Internet, mentre alcuni apparecchi possono essere collegati direttamente al cloud. I servizi in esecuzione nel cloud raccolgono dati dagli edge device, li analizzano e li forniscono all'utente finale [29].

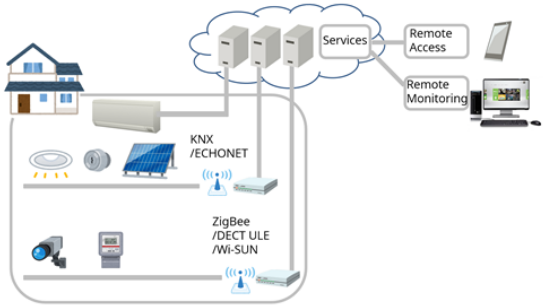


Figura 2.5: Caso d'uso settore consumer: Smart Home [29]

Settore industriale

Il caso d'uso in questo settore è un esempio di Smart Factory, come mostrato in figura 2.6. Un edge device industriale raccoglie dati selezionati da vari controller e li rende disponibili a un servizio di backend cloud, ad esempio per il monitoraggio remoto tramite una dashboard o li analizza per la manutenzione preventiva.

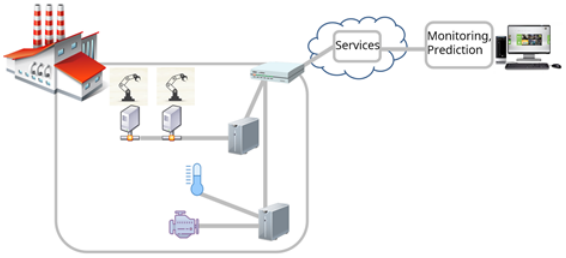


Figura 2.6: Caso d'uso settore industriale: Smart Factory [29]

Trasporto e logistica

Il monitoraggio dei veicoli, dei costi del carburante, delle esigenze di manutenzione e degli incarichi aiuta a ottimizzare il pieno utilizzo della flotta di veicoli. Le spedizioni possono essere tracciate per garantire qualità e condizioni costanti delle merci trasportate. Ciò è particolarmente utile per garantire l'integrità della catena del freddo dai magazzini ai camion frigoriferi fino alla consegna. Il monitoraggio e la gestione centralizzata delle scorte nei magazzini possono prevenire situazioni di esaurimento.

Agricoltura

Il monitoraggio delle condizioni del suolo e la creazione di piani ottimali per l'irrigazione, la concimazione e il monitoraggio delle condizioni dei prodotti ottimizzano la qualità e la produzione dei prodotti agricoli.

Assistenza sanitaria

La raccolta dei dati e l'analisi dei dati degli studi clinici aiutano a ottenere informazioni dettagliate su nuove aree. Il monitoraggio remoto dei pazienti riduce il rischio di situazioni critiche non rilevate per gli anziani e i pazienti dopo il ricovero.

Smart Cities

Il monitoraggio di ponti, dighe, argini, canali per controllarne lo stato materiale e il deterioramento previene danni significativi. Il monitoraggio delle autostrade e la fornitura di segnaletica adeguata garantiscono un flusso di traffico ottimizzato. Lo Smart Parking ottimizza e tiene traccia dell'utilizzo e della disponibilità dei parcheggi e automatizza la fatturazione/prenotazione. Il controllo intelligente delle luci stradali basato sul rilevamento della presenza di pedoni o automobili, le previsioni del tempo, ecc. riducono i costi. I contenitori dei rifiuti possono essere monitorati per ottimizzare la gestione dei rifiuti e il percorso di raccolta dei rifiuti.

2.3.2 Requisiti

2.3.3 Requisiti funzionali

Nei requisiti funzionali vengono definite le proprietà richieste che un'architettura WoT deve avere [29].

Principi comuni

- l'architettura WoT dovrebbe consentire l'interazione reciproca di diversi ecosistemi utilizzando la tecnologia Web;
- l'architettura WoT dovrebbe essere basata sull'architettura Web utilizzando le API RESTful;
- l'architettura WoT dovrebbe consentire di utilizzare più formati di payload comunemente usati nel Web;
- L'architettura WoT deve abilitare diverse architetture di dispositivi e non deve forzare un'implementazione client o server dei componenti di sistema;
- **flessibilità**: esiste un'ampia varietà di configurazioni di dispositivi fisici per le implementazioni WoT. L'architettura astratta WoT dovrebbe poter essere mappata e coprire tutte le variazioni;
- **compatibilità**: esistono già molte soluzioni IoT esistenti e attività di standardizzazione IoT in corso in molti settori aziendali. Il WoT dovrebbe fornire un ponte tra queste soluzioni IoT esistenti e in via di sviluppo e la tecnologia Web basata sui concetti WoT. Il WoT dovrebbe essere compatibile con le soluzioni IoT esistenti e gli standard attuali;
- **scalabilità**: il WoT deve essere in grado di adattarsi a soluzioni IoT che incorporano da migliaia a milioni di dispositivi. Questi dispositivi possono offrire le stesse capacità anche se sono stati creati da produttori diversi;

- **interoperabilità**: il WoT deve fornire l'interoperabilità tra i produttori di dispositivi e cloud. Deve essere possibile prendere un dispositivo abilitato WoT e collegarlo con un servizio cloud di diversi produttori pronto all'uso.

Funzionalità di una Thing

L'architettura WoT dovrebbe consentire alle Things di avere funzionalità come:

- lettura e aggiornamento delle informazioni di stato;
- subscribing e unsubscribing alle notifiche di modifiche delle informazioni sullo stato della Thing;
- invocando funzioni con parametri di input e output che causerebbero determinate attivazioni o calcoli.

Ricerca e scoperta

- l'architettura WoT dovrebbe consentire ai Client di conoscere gli attributi, le funzionalità e i punti di accesso della Thing, prima di accedere alla Thing stessa;
- l'architettura WoT dovrebbe consentire ai Client di cercare le Things in base agli attributi e alle funzionalità della Thing;
- l'architettura WoT dovrebbe consentire la ricerca semantica delle Things fornendo le funzionalità richieste basate su un vocabolario unificato, indipendentemente dalla denominazione delle funzionalità.

Meccanismo di descrizione

L'architettura WoT dovrebbe supportare un meccanismo di descrizione comune che consenta di descrivere le Things e le loro funzioni, tali descrizioni dovrebbero essere non solo leggibili dall'uomo, ma anche leggibili dalla

macchina, dovrebbero consentire l'annotazione semantica della sua struttura e dei contenuti descritti e dovrebbero poter essere scambiati utilizzando più formati comunemente usati nel web.

Descrizione degli attributi

L'architettura WoT dovrebbe consentire di descrivere gli attributi della Thing come: nome, versione delle specifiche, formato, descrizione, collegamenti ad altre Things correlate e informazioni sui metadati. Tali descrizioni dovrebbero supportare l'internazionalizzazione.

Network

L'architettura WoT dovrebbe supportare più protocolli Web comunemente usati. Tali protocolli includono: protocolli comunemente usati su Internet e protocolli comunemente usati nella rete locale. L'architettura WoT dovrebbe consentire l'utilizzo di più protocolli Web per accedere alla stessa funzionalità e dovrebbe consentire l'utilizzo di una combinazione di più protocolli per le funzionalità della stessa Thing (ad es. HTTP e WebSocket).

Deployment

L'architettura WoT dovrebbe supportare un'ampia varietà di funzionalità come edge device con limitazioni delle risorse e virtual Thing sul cloud, basati sullo stesso modello. Dovrebbe supportare più livelli di gerarchia delle Things con entità intermedie come gateway o proxy e dovrebbe supportare l'accesso alle Things nella rete locale dall'esterno della rete locale (Internet o un'altra rete locale) considerando la traduzione dell'indirizzo di rete.

Applicazione

L'architettura WoT dovrebbe consentire la descrizione di applicazioni per un'ampia varietà di Things come dispositivi edge, gateway, cloud e dispositivi UI/UX utilizzando la tecnologia standard del Web.

Legacy adoption

L'architettura WoT dovrebbe consentire la mappatura dei protocolli IP e non IP legacy sui protocolli Web, supportando varie topologie, in cui tali protocolli legacy vengono terminati e tradotti. Inoltre, dovrebbe consentire un uso trasparente dei protocolli IP esistenti senza traduzione, che seguono l'architettura RESTful e infine non deve imporre ruoli Client o Server su dispositivi e servizi. Un dispositivo IoT può essere un Client o un Server, o entrambi, a seconda dell'architettura del sistema, lo stesso vale per i servizi edge e cloud.

2.3.4 Requisiti tecnici

I Pattern comuni dei requisiti funzionali appena descritti definiscono l'architettura astratta del WoT, questa sezione descrive i requisiti tecnici derivati dall'architettura astratta [29].

Componenti del Web of Things e dell'architettura Web of Things

I casi d'uso aiutano a identificare i componenti di base come i dispositivi e le applicazioni, che accedono e controllano tali dispositivi, proxy (ad esempio gateway ed edge device) che si trovano tra i dispositivi. Un componente aggiuntivo utile in alcuni casi d'uso è la directory, che aiuta al rilevamento del servizio. Tali componenti sono collegati a Internet o alle reti locali, fabbriche o altre strutture. Tutti i componenti coinvolti possono essere connessi a una singola rete o in generale i componenti possono essere distribuiti su più reti.

Dispositivi

Nei dispositivi l'accesso avviene tramite una descrizione delle loro funzioni e delle loro interfacce. Questa descrizione è chiamata Thing Description (TD). Una Thing Description include metadati generali sul dispositivo, modelli informativi che rappresentano funzioni, descrizione del protocollo di trasporto per operare su modelli informativi e informazioni sulla sicurezza.

I metadati generali contengono identificatori del dispositivo (URI), informazioni sul dispositivo come numero di serie, data di produzione, posizione e altre informazioni leggibili dall'uomo. I modelli informativi definiscono gli attributi del dispositivo e rappresentano le impostazioni interne del dispositivo, la funzionalità di controllo e la funzionalità di notifica. I dispositivi che hanno la stessa funzionalità hanno lo stesso modello di informazioni indipendentemente dai protocolli di trasporto utilizzati. Poiché molti sistemi basati sull'architettura Web of Things stanno attraversando domini di sistema differenti, i vocabolari e i metadati (ad esempio, ontologie) utilizzati nei modelli informativi dovrebbero essere comunemente compresi dalle parti coinvolte. Oltre all'architettura REST, sono supportati anche i meccanismi PubSub. Le informazioni sulla sicurezza includono descrizioni sull'autenticazione, l'autorizzazione e le comunicazioni sicure. I dispositivi sono tenuti a inserire le TD al loro interno o in posizioni esterne ai dispositivi e a rendere accessibili le TD in modo che altri componenti possano trovarli e accedervi.

Applicazioni

Le applicazioni devono essere in grado di generare e utilizzare interfacce di rete e programmi basati su TD e devono essere in grado di ottenere queste descrizioni attraverso la rete, quindi devono essere in grado di condurre operazioni di ricerca e acquisire le descrizioni necessarie sulla rete.

Digital twins

I Digital Twin devono generare interfacce di programma internamente basate su TD e rappresentare virtual device utilizzando tali interfacce di programma. Un "twin" deve produrre una TD per il virtual device e renderlo disponibile esternamente. Gli identificatori dei virtual device devono essere assegnati in quanto sono diversi dai dispositivi originali, ciò assicura che i virtual device e i dispositivi originali siano chiaramente riconosciuti come entità separate. I meccanismi e le impostazioni di trasporto e sicurezza dei virtual device possono essere diversi dai dispositivi originali, se necessario.

I virtual device devono avere descrizioni fornite direttamente dal "twin" o averle disponibili in posizioni accessibili dall'esterno. In entrambi i casi è necessario rendere disponibili le descrizioni in modo che altri componenti possano trovare e utilizzare i dispositivi ad essi associati.

Discovery

Affinché le TD dei dispositivi e dei virtual device siano accessibili da dispositivi, applicazioni e twins, è necessario un modo comune per condividere le TD. Le directory possono soddisfare questo requisito fornendo funzionalità per consentire ai dispositivi e ai twins automaticamente o agli utenti di registrare manualmente le descrizioni. Le descrizioni dei dispositivi e dei virtual device devono essere ricercabili da entità esterne.

Sicurezza

Le informazioni sulla sicurezza relative ai dispositivi e ai virtual device devono essere descritte nelle descrizioni dei dispositivi. Ciò include le informazioni per l'autenticazione/autorizzazione e la crittografia. L'architettura WoT dovrebbe supportare più meccanismi di sicurezza comunemente usati nel web, come Basic, Digest, Bearer e OAuth2.0.

Accessibilità

Il Web of Things si rivolge principalmente alla comunicazione machine to machine. Gli esseri umani coinvolti sono solitamente sviluppatori che integrano le Things nelle applicazioni. Gli utenti finali dovranno confrontarsi con i front-end delle applicazioni o le interfacce utente fisiche fornite dai dispositivi stessi. Entrambi non rientrano nell'ambito delle specifiche WoT del W3C. Data l'attenzione all'IoT invece che agli utenti, l'accessibilità non è un requisito diretto e quindi non è trattata all'interno di questa specifica. C'è, tuttavia, un aspetto interessante sull'accessibilità: soddisfare i requisiti di cui sopra consente alle macchine di comprendere le API dei dispositivi

collegati alla rete. Questo può essere utilizzato da strumenti di accessibilità per fornire interfacce utente di diverse modalità, rimuovendo così le barriere all'utilizzo di dispositivi fisici e ad applicazioni IoT.

2.3.5 Architettura WoT

Per affrontare i casi d'uso descritti precedentemente e soddisfare i requisiti appena descritti, il Web of Things si basa sul concetto di Web Things, solitamente chiamato semplicemente Things, che può essere utilizzato dai Consumer, cioè un'entità in grado di elaborare le descrizioni delle Things WoT (incluso il suo formato di rappresentazione basato su JSON) e interagire con le Things (cioè consumare le Things). In questa sezione saranno fornite le normative per definire l'architettura complessiva del Web of Things del W3C [29].

2.3.6 Panoramica generale

Una Thing è l'astrazione di un'entità fisica o virtuale (ad esempio, un dispositivo o una stanza) ed è descritta da metadati standardizzati. Nel W3C WoT, i metadati descrittivi devono essere una WoT Thing Description. I Consumer devono essere in grado di analizzare ed elaborare il formato di rappresentazione della TD, che si basa su JSON, in figura 2.7 è possibile vedere una rappresentazione di interazione. Il formato può essere elaborato tramite le classiche librerie JSON o un processore JSON-LD, poiché il modello informativo sottostante è basato su grafici e la sua serializzazione è compatibile con JSON-LD. L'uso di un processore JSON-LD per l'elaborazione di un TD consente inoltre l'elaborazione semantica (inclusa la trasformazione in triple RDF), l'inferenza semantica e l'esecuzione di compiti dati in base a termini ontologici, questo rende i consumer più autonomi. Una TD è specifica dell'istanza, descrive una singola Thing, non tipi di Things, ed è la rappresentazione testuale esterna (Web) predefinita di una Thing. Per essere una Thing deve essere disponibile almeno una rappresentazione della TD.

La WoT Thing Description è un formato di rappresentazione standardizzato e comprensibile dalla macchina che consente ai consumer di scoprire e interpretare le capacità di una Thing (attraverso le annotazioni semantiche) e di adattarsi a diverse implementazioni (ad esempio, protocolli o strutture di dati diversi) quando interagiscono con una Thing, consentendo così l'interoperabilità tra diverse piattaforme IoT, ovvero ecosistemi e standard diversi [29].

Figura 2.7: Interazione con la Thing: Consumer [29]

Una Thing può anche essere l'astrazione di un'entità virtuale. Un'entità virtuale è la composizione di una o più Things (ad esempio, una stanza composta da più sensori e attuatori). Un'opzione per la composizione consiste nel fornire una descrizione WoT unica che contenga diverse funzionalità per l'entità virtuale. Nei casi in cui la composizione è piuttosto complessa, la TD può avere link a sub-Things gerarchiche all'interno della composizione. La TD principale funge da punto di ingresso, contiene solo metadati e ciò che può fare. Ciò consente il raggruppamento di alcuni aspetti di Things più complesse. Il linking non si applica solo alle Things gerarchiche, ma anche alle relazioni tra le Things e altre risorse in generale. I tipi di relazione di link esprimono come le Things si relazionano, ad esempio, un interruttore che controlla una luce o una stanza monitorata da un sensore di movimento. Altre risorse relative a una Thing possono essere manuali, cataloghi di pezzi di ricambio, file CAD, un'interfaccia utente grafica o qualsiasi altro documento sul Web. Nel complesso, il linking Web tra le Things rende il WoT navigabile, sia per gli esseri umani che per le macchine. Ciò può essere ulteriormente facilitato fornendo directory Thing che gestiscono un catalogo di Things disponibili, di solito memorizzando nella cache la rappresentazione della TD. In sintesi, le descrizioni delle Things del WoT possono collegarsi

ad altre Things e ad altre risorse sul Web per formare una rete di Things, in figura 2.8 è possibile visualizzare un modello che rappresenta una rete di Things.

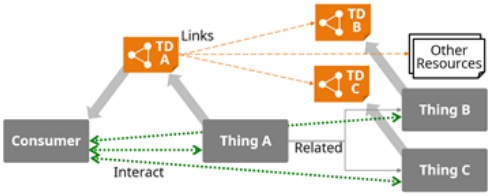


Figura 2.8: Interazione con la Thing: Linked Things [29]

Le Things devono essere esposte su interfaccia di rete per realizzare l'interfaccia WoT di una Thing. Un esempio è un server HTTP in esecuzione su un dispositivo embedded con sensori e attuatori che interfacciano l'entità fisica dietro l'astrazione della Thing. Tuttavia, W3C WoT non impone dove sono ospitate le Things, può trovarsi direttamente sul dispositivo IoT, o su un dispositivo edge come un gateway o il cloud. Una tipica sfida di distribuzione è uno scenario in cui le reti locali non sono raggiungibili da Internet, in genere a causa di NAT (Network Address Translation) IPv4 o dispositivi firewall. Per porre rimedio a questa situazione, W3C WoT consente Intermediari tra Things e Consumer. Gli intermediari possono fungere da proxy per le Things, laddove l'intermediario ha una descrizione della Thing WoT simile alla Thing originale, ma che punta all'interfaccia WoT fornita dall'intermediario. Gli intermediari possono anche aumentare le Things esistenti con capacità aggiuntive o comporre una nuova Thing da più Things disponibili, formando così un'entità virtuale. Per i Consumer, gli intermediari sembrano Things, poiché possiedono descrizioni di Things WoT e forniscono un'interfaccia WoT, quindi potrebbero essere indistinguibili dalle Things in un'architettura di sistema a più livelli come il Web. Un identificatore nella descrizione della Thing WoT deve consentire la correlazione di più TD che rappresentano la stessa Thing originale o in definitiva un'entità fisica unica. In figura 2.9 è possibile vedere il modello di un Intermediary.

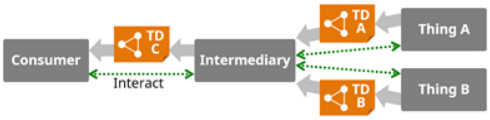


Figura 2.9: Interazione con la Thing: Intermediary [29]

Un altro rimedio per le reti locali è vincolare l'interfaccia WoT a un protocollo che stabilisce la connessione dalla Things dall'interno della rete locale a un Consumer raggiungibile pubblicamente. Le Things possono essere raggruppate insieme a un Consumer per consentire l'interazione da Thing a Thing. Di solito, il comportamento del Consumer è incorporato nel componente software, che implementa anche il comportamento della Thing. La configurazione del comportamento del Consumer può essere esposta attraverso la Thing. I concetti del W3C WoT sono applicabili a tutti i livelli rilevanti per le applicazioni IoT: il livello del dispositivo, il livello edge e il livello del cloud. Ciò promuove interfacce e API comuni tra i diversi livelli e consente vari modelli di integrazione come Thing-to-Thing, Thing-to-Gateway, Thing-to-Cloud, Gateway-to-Cloud e persino la federazione del cloud, ovvero l'interconnessione del cloud in ambienti di due o più fornitori di servizi, per applicazioni IoT. La figura 2.10 fornisce una panoramica di come i concetti WoT introdotti sopra possono essere applicati e combinati per affrontare i casi d'uso descritti precedentemente.

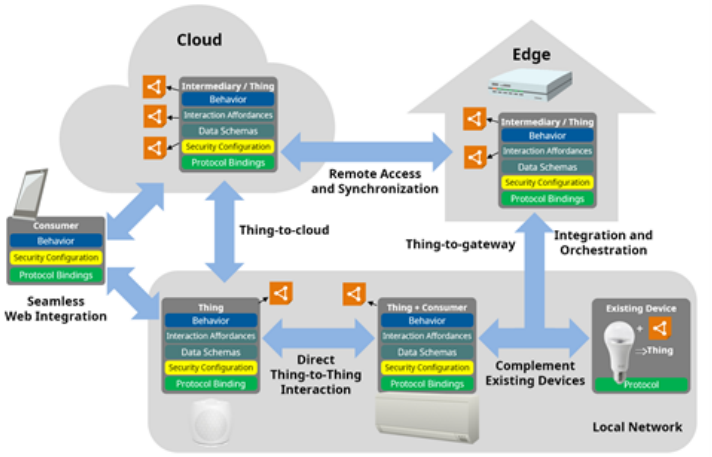


Figura 2.10: Esempio di architettura astratta del W3C WoT [29]

2.3.7 Affordances

Un aspetto centrale nel W3C WoT è la fornitura di metadati comprensibili dalla macchina (cioè, WoT Thing Descriptions). Idealmente, tali metadati sono autodescrittivi, in modo che i Consumer siano in grado di identificare quali funzionalità fornisce una Thing e come utilizzare le proprietà fornite. Questa descrittività di una Thing è il concetto di affordances. In questo contesto, un'affordance è la descrizione di un collegamento ipertestuale che suggerisce ai client Web come navigare ed eventualmente come agire sulla risorsa collegata. Pertanto, i collegamenti forniscono vantaggi per la navigazione. Il Web of Things definisce Interaction Affordances come metadati di una Thing che mostra e descrive le possibili scelte ai Consumer, suggerendo così come i Consumer possono interagire con la Thing. Un Interaction Affordance generale è la navigazione, che si attiva seguendo un link, consentendo così ai Consumer di navigare nel Web of Things. Il modello di interazione definisce altri tre tipi di interazione nel W3C WoT: Properties, Actions ed Event, che vedremo di seguito.

2.3.8 Web Thing

Una Web Thing ha quattro aspetti architetturali di interesse: il suo behaviour, le sue Interaction Affordances, la sua configurazione di sicurezza e i suoi Protocol Binding, come illustrato nella Figura 2.11. Il behaviour di una Thing include sia il behaviour autonomo che le Interaction Affordances. Le Interaction Affordances forniscono un modello di come i Consumer possono interagire con la Thing attraverso operazioni astratte, ma senza riferimento a uno specifico protocollo di rete o codifica dei dati. Il Protocol Binding aggiunge dettagli necessari per mappare ogni Interaction Affordance a messaggi concreti di un determinato protocollo. In generale, diversi protocolli possono essere utilizzati per supportare diversi sottoinsiemi di interazione, anche all'interno di una singola Thing. L'aspetto di configurazione della sicurezza di una Thing rappresenta i meccanismi utilizzati per controllare l'accesso alle Interaction Affordances e la gestione dei relativi metadati.

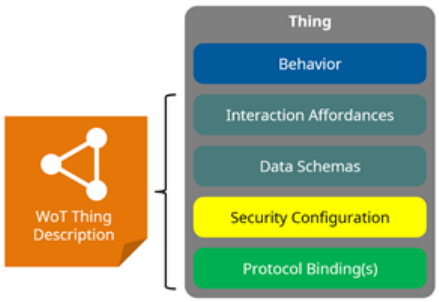


Figura 2.11: Aspetto architetturale di una Thing [29]

2.3.9 Modello di interazione

Il modello di interazione del W3C WoT introduce un'astrazione che formalizza la mappatura dall'intento dell'applicazione alle operazioni concrete del protocollo. Le Things possono offrire altri tre tipi di vantaggi di interazione definiti da questa specifica: Properties, Actions ed Events.

Properties

Una Properties è un Interaction Affordance che espone lo stato della Thing. Lo stato esposto da una Property deve essere recuperabile (leggibile). Facoltativamente, lo stato esposto da una Property può essere aggiornato (scrivibile). Le Things possono scegliere di rendere osservabili le Properties inviando il nuovo stato dopo un cambiamento. Lo stato di write-only deve essere aggiornato tramite un'Action. Se i dati non sono completamente specificati dal Protocol Binding utilizzato le Properties possono contenere uno schema di dati per lo stato esposto.

Actions

Un'Actions è un'Interaction Affordance che permette di invocare una funzione della Thing. Un'Action può manipolare uno stato (Properties) che non è direttamente esposto, manipolare più Properties alla volta o manipolare Properties in base alla logica interna. Invocare un'Action può anche innescare un processo sulla Thing che manipola lo stato nel tempo. Se i dati non sono completamente specificati dal Protocol Binding utilizzato, le Actions possono contenere schemi di dati per parametri di input e risultati di output opzionali.

Events

Un Event è un Interaction Affordance che descrive un evento che invia i dati in modo asincrono dalla Thing al Consumer. Qui non vengono comunicate transizioni di stato, ma di stato (cioè eventi). Gli Events possono essere attivati da condizioni che non sono esposte come Properties. Se i dati non sono completamente specificati dal Protocol Binding utilizzato, gli Events possono contenere schemi di dati per i dati dell'Event e possibili messaggi di controllo della sottoscrizione.

2.3.10 Protocol Binding

Un Protocol Binding è la mappatura da un Interaction Affordance a messaggi concreti di un protocollo specifico come HTTP, CoAP o MQTT. Informa il Consumer su come attivare l'Interaction Affordance attraverso un'interfaccia di rete. Le associazioni del protocollo seguono il vincolo dell'interfaccia REST per supportare l'interoperabilità. Pertanto, non tutti i protocolli di comunicazione sono idonei per implementare Protocol Bindings per W3C WoT.

2.3.11 Servient

È stata descritta l'architettura WoT in termini di componenti astratte dell'architettura WoT come Things, Consumer e Intermediary. Quando questi componenti astratti dell'architettura WoT vengono implementati come stack software per assumere un ruolo specifico nell'architettura WoT, tali stack software vengono chiamati Servients. Di seguito verranno illustrati i diagrammi di come i Servient interagiscono per creare sistemi basati sull'architettura WoT [29].

Thing implementata come Servient

Uno stack software Servient contiene una rappresentazione di una Thing chiamata Exposed Thing e rende la sua interfaccia WoT disponibile ai Consumer della Thing (figura 2.12).

Figura 2.12: Thing implementata come Servient [29]

Consumer implementato come Servient

I Consumer sono implementati dai Servient in quanto devono elaborare la Thing Description e devono avere uno stack di protocollo che può essere configurato tramite le informazioni di Protocol Binding contenute nella TD. Nel Consumer, uno stack software Servient fornisce una rappresentazione di una Thing chiamata Consumed Thing e la rende disponibile per le applicazioni in esecuzione sul Servient che devono elaborare TD per interagire con le Things (figura 2.13).

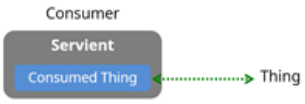


Figura 2.13: Consumer implementato Servient [29]

Intermediary implementato come Servient

Un Intermediary si trova tra una Thing e i suoi Consumer, svolgendo i ruoli sia di Consumer (per la Thing) che di Thing (per i Consumer). In un Intermediary, uno stack software Servient contiene le rappresentazioni sia di un Consumer (Consumed Thing) che di una Thing (Exposed Thing), (figura 2.14).

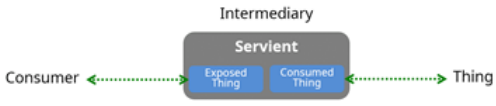


Figura 2.14: Intermediary implementato come Servient [29]

La comunicazione tra Servient può avvenire in due diverse modalità:

Comunicazione diretta

La comunicazione diretta si applica quando entrambi i Servient utilizzano gli stessi protocolli di rete e sono reciprocamente accessibili (figura 2.15).

Figura 2.15: Servient: comunicazione diretta [29]

Comunicazione indiretta

La comunicazione indiretta si applica quando i Servient non utilizzano gli stessi protocolli di rete o sono su reti diverse e sono messi in comunicazione tramite gli Intermediary (figura 2.16).

Figura 2.16: Servient: comunicazione indiretta [29]

2.3.12 WoT Building Blocks

I WoT Building Blocks consentono l'implementazione di sistemi conformi all'architettura astratta WoT. Questa sezione fornisce una panoramica e un riepilogo di ciò che è stato discusso precedentemente.

I WoT Building Blocks supportano tutti gli strati dell'architettura finora discussa. I singoli Building Blocks sono mostrati nel contesto di una Thing astratta in Figura 2.17. La figura 2.17 illustra il rapporto tra i Building Blocks e i principali aspetti dell'architettura di una Thing.

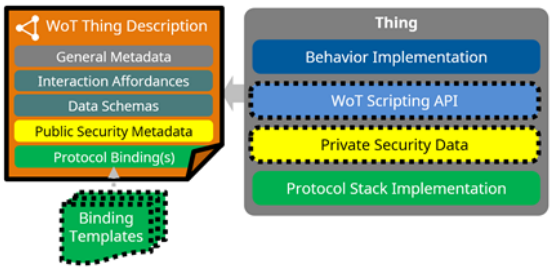


Figura 2.17: Relazione tra i WoT Building Blocks e gli strati architetturali di una Thing [29]

2.3.13 WoT Thing Description

La Thing Description definisce un modello informativo basato su un vocabolario semantico e una rappresentazione serializzata basata su JSON. Le TD forniscono metadati avanzati per le Things in un modo che sia leggibile dall'uomo e comprensibile dalla macchina. Sia il modello informativo che il formato di rappresentazione delle TD sono allineati con JSON Linked Data in modo che oltre all'elaborazione JSON grezza, le implementazioni possano scegliere di utilizzare JSON-LD e database di grafi per l'elaborazione dei metadati. Una Thing Description descrive le istanze di Thing con metadati generali come nome, ID, descrizioni e può anche fornire metadati di relazione tramite collegamenti (Link) a oggetti correlati o altri documenti. Le TD contengono anche le Interaction Affordance basate sul modello di interazione definito precedentemente. La TD può essere vista come index.html per le Things in quanto fornisce il punto di ingresso per conoscere i servizi forniti e le relative risorse. La TD viene creata e/o ospitata dalla Thing stessa e recuperata al momento della scoperta. Tuttavia, può anche essere ospitata esternamente quando una Thing ha limitazioni di risorse (ad es. spazio di memoria limitato, potenza limitata) o quando un dispositivo esistente viene adattato per diventare parte del WoT. La Thing Description promuove l'interoperabilità in due modi: in primo luogo, le TD consentono la comunicazione

machine to machine nel WoT, in secondo luogo, le TD possono fungere da formato comune e uniforme per gli sviluppatori per documentare e recuperare tutti i dettagli necessari per creare applicazioni in grado di accedere ai dispositivi IoT e utilizzare i loro dati.

2.3.14 WoT Binding Templates

L'IoT utilizza una varietà di protocolli per l'accesso ai dispositivi, poiché nessun protocollo singolo è appropriato in tutti i contesti, una sfida centrale per il Web of Things è consentire interazioni con le diverse piattaforme IoT (ad es. OCF, oneM2M, OMA LWM2M, OPC UA) e dispositivi che non seguono alcuno standard particolare ma forniscono un'interfaccia idonea su un protocollo di rete. WoT affronta questa varietà attraverso i Protocol Bindings, che devono soddisfare una serie di vincoli.

2.3.15 WoT Scripting API

Il WoT Scripting API è un elemento opzionale del W3C WoT che facilita lo sviluppo di applicazioni IoT fornendo un'API basata su ECMAScript simile alle API del browser Web, integrando un sistema di runtime scripting nel WoT Runtime, il WoT Scripting API consente di utilizzare script di applicazioni che definiscono il comportamento di Things, Consumer e Intermediary. Il WoT Scripting API consente l'implementazione della logica del dispositivo mediante script riutilizzabili eseguiti in un sistema di runtime per applicazioni IoT non dissimile da quello di un browser Web e mira a migliorare la produttività e ridurre i costi di integrazione.

2.3.16 WoT Security and Privacy Guidelines

Nell'architettura WoT, la sicurezza è supportata da alcune funzionalità esplicite, come il supporto per i metadati di sicurezza nelle TD. La specifica per ogni building block forniscono un'indicazione sulla sicurezza e sulla privacy di quel building block.

Capitolo 3

Progettazione

3.1 Obiettivi

L'obiettivo di questa tesi è realizzare un framework multipiattaforma in grado di creare, modificare, compilare e fare l'upload di codice W3C WoT compliant per dispositivi embedded. In particolare, tale framework è un'applicazione Web in grado di creare codice sorgente per dispositivi embedded Espressif ESP32 a partire da un form HTML; inoltre consente all'utente di modificare, tramite editor, il codice generato e di visualizzare tali modifiche tramite form HTML e viceversa; infine, scarica le librerie necessarie, compila e fa l'upload del codice sui dispositivi. Il software crea codice sorgente per Thing e Consumer. Per la creazione della Thing consente di creare Properties, Actions ed Events. Per la creazione del Consumer è in grado di leggere una Thing Description, mostrare le Interaction Affordances e abilitare i Protocols Binding disponibili. Infine, per la creazione del Consumer, oltre ad abilitare i Protocols Binding il software dà la possibilità al Consumer di essere anche una Thing, attraverso la creazione di Properties, Actions ed Events. Tramite questo software gli sviluppatori possono progettare, compilare e installare applicazioni WoT W3C compliant su sistemi embedded.

3.2 Stato dell'arte

Il presente progetto rappresenta una nuova versione del lavoro di tesi svolto da Federico Rachelli [25]. A differenza del lavoro eseguito in precedenza, questo progetto ha differenti obiettivi e requisiti ed è stato interamente riscritto utilizzando differenti tecnologie. Il risultato del lavoro di tesi svolto in precedenza consiste in un'applicazione che consente all'utente di creare una Thing, di compilare lo sketch e di fare l'upload sul dispositivo embedded. Tale progetto ha dei limiti: non è possibile modificare il codice sorgente dall'applicazione, non è multipiattaforma e non consente di creare la parte Consumer. Nel mio lavoro di tesi riscrivo con nuove tecnologie l'obiettivo del progetto del collega aggiungendo nuove funzionalità e togliendo i limiti precedentemente descritti. In questo capitolo descriverò gli obiettivi, i requisiti e l'architettura del mio lavoro di tesi.

3.3 Requisiti funzionali

Di seguito sono elencati i requisiti che il software deve fornire:

- **Multipiattaforma**
  Il software deve poter girare su più sistemi operativi.

- **Generazione sketch**
  Il software deve essere in grado di creare sketch per dispositivi embedded.

- **Visualizzazione codice**
  L'utente deve avere la possibilità di visualizzare il codice generato dal software.

- **Modifica codice tramite editor**
  L'utente deve avere la possibilità di modificare il codice generato dal software tramite code editor.

- **Modifica codice tramite form**
  L'utente deve avere la possibilità di modificare il codice tramite form HTML.

- **Aggiornamento codice**
  L'utente, indipendentemente dal tipo di modifica, deve vedere il codice aggiornato in entrambi i tipi di modifica.

- **Compilazione e upload dello sketch**
  L'utente deve poter compilare il codice generato e fare l'upolad sul dispositivo embedded e deve poter visualizzare gli errori che si verificano in fase di compilazione e upload.

- **Visualizzazioni errori**
  L'utente deve poter visualizzare gli errori generati in fase di compilazione o upload dello sketch.

- **Eliminazione Progetto**
  L'utente deve avere la possibilità di cancellare il progetto creato e tutti i suoi file o record creati per quel progetto.

I requisiti funzionali appena elencati devono essere sviluppati per funzionare in contesto Thing e Consumer. Di seguito vengono dettagliati i requisiti funzionali per i due contesti:

Thing

- **Creazione della Thing Description**
  L'utente deve poter creare la Thing Description fornendo tutti i dati della Thing che intende creare.

- **Scelta dei Binding Templates**
  L'utente deve poter scegliere uno o più Binding Templates disponibili per le Interaction Affordances.

- **Modifica del codice sorgente**
  Dopo aver generato il codice sorgente a partire dal form HTML l'utente deve poter modificare il codice generato tramite editor e deve poter visualizzare tali modifiche anche nel form; inoltre, dopo la creazione della Thing deve poter modificare la TD e visualizzare tali modifiche anche nell'editor.

- **Compilazione e upload dello sketch**
  L'utente deve poter compilare il codice generato e fare l'upolad sul dispositivo embedded e deve poter visualizzare eventuali errori che si verificano in fase di compilazione e upload.

Consumer

- **Lettura della Thing Description**
  L'utente deve poter inserire l'URL di una TD esposta nella rete oppure poter inserire manualmente una TD e scegliere le Interaction Affordances da consumare.

- **Scelta Interaction Affordances**
  L'utente deve avere la possibilità di scegliere le Interaction Affordances rilevate dalla lettura della TD.

- **Scelta Protocol Binding**
  L'utente deve avere la possibilità di scegliere le Interaction Affordances e i relativi Protocol Binding della TD.

- **Modifica del codice sorgente**
  Dopo aver generato il codice sorgente a partire dalla lettura della TD, l'utente deve poter modificare il codice generato tramite editor e deve poter visualizzare tali modifiche anche nel form.

- **Compilazione e upload dello sketch**
  L'utente deve poter compilare il codice generato, fare l'upolad sul di-

spositivo embedded e visualizzare gli errori che si verificano in fase di compilazione e upload.

Inoltre, il software dovrà avere i seguenti requisiti tecnici:

- **Esposizione Thing Description**
  La Thing Description deve poter essere visualizzata da altri dispositivi, quindi la Thing deve potersi connettere alla rete in modo da avere un URL raggiungibile per poter essere consumata.
- **Protocol Bindings**
  La Thing deve implementare i Binding Templates per poter comunicare attraversi i protocolli HTTP, WebSocket, CoAP ed MQTT.

3.4 Architettura

In questa sezione viene presentata l'architettura implementata per questo progetto, nella figura 3.1 si possono visualizzare graficamente le componenti dell'architettura e le loro interazioni.

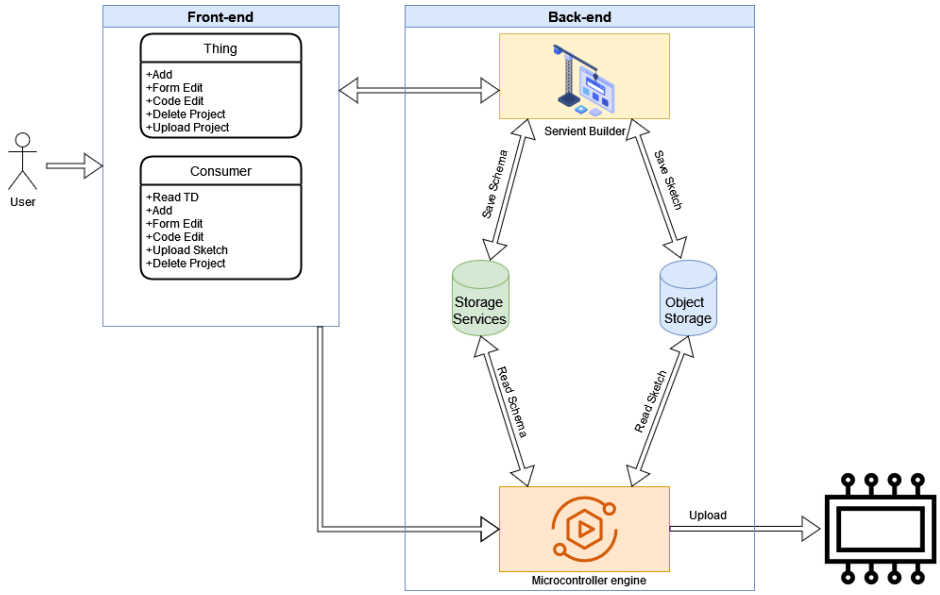


Figura 3.1: Architettura progetto

Di seguito verranno dettagliati i vari componenti dell'architettura e le loro funzionalità:

3.5 Front-end

L'interfaccia grafica è stata implementata come applicazione Web ed è la componente che l'utente utilizza per creare e modificare le Thing o i Consumer, per compilare e fare l'upload sulla board e per eliminare i progetti. Tramite questa interfaccia l'utente ha la possibilità di creare, modificare, compilare e flashare una Thing e/o un Consumer. I form HTML presenti nel front-end sono stati creati tramite una libreria javascript, discussa nel Capitolo 4, che consente di creare degli schemi in JSON e convertirli in form HTML.
La parte front-end dell'architettura consente di eseguire le seguenti azioni:

3.5.1 Creazione Thing

L'interfaccia grafica della creazione della Thing è suddivisa in quattro tab HTML:

Build

In questa tab vengono inserite le informazioni generali della Thing, nello specifico l'utente può inserire:

- dati relativi alla TD:
  - nome Thing;
  - versione TD W3C;
  - parametri sicurezza.
- dati connessione SSID e password WiFi;
- librerie Arduino.

Interaction Affordances

Questa sezione è suddivisa in tre tabs e si occupa di rilevare tutte le informazioni necessarie alla creazione della Thing; le tre tabs sono così suddivise:

- Properties
- Actions
- Events

non c'è limite alla creazione delle Interaction Affordances e ognuna di esse ha un nome e quattro possibili Protocol Binding:

- HTTP
- WebSocket
- CoAP
- MQTT

inoltre, è possibile inserire i tipi di interazione che le Interaction Affordances devono avere, come ad esempio lettura e/o scrittura.

3.5.2 Modifica Thing

La modifica della Thing può avvenire in due modi diversi:

- tramite form HTML
- tramite editor

La modifica tramite form HTML si presenta uguale al form utilizzato in fase di creazione della Thing, la differenza è che in fase di modifica i campi sono popolati da ciò che è stato inserito in fase di creazione. Qualsiasi elemento può essere aggiornato, cancellato o aggiunto. La modifca tramite editor consente all'utente di visualizzare lo sketch generato e dà la possibilità

di modificare, aggiungere o cancellare qualsiasi snippet di codice all'interno. La caratteristica fondamentale di questi due tipi di modifiche è che, indipendentemente da quale tipologia di modifica si scelga, il risultato della modifica viene replicato per entrambe le tipologie. Per esempio, se si aggiunge una nuova Properties tramite editor si vedrà la nuova Properties anche nella modifica tramite form HTML e viceversa. Le modifiche effettuate tramite editor che si vedranno anche nella modifica tramite form HTML riguardano solo le Interaction Affordances, tutte le altre modifiche saranno comunque salvate nello sketch ma non visibili nel form HTML in quanto non riguardano la Thing Description.

3.5.3 Creazione Consumer

A differenza della creazione della Thing, la creazione del Consumer ha componenti differenti. Di seguito viene dettagliata l'architettura della creazione del Consumer.

Lettura Thing Description

In questa sezione viene mostrato un form HTML che offre due modi possibili di lettura della TD:

- tramite url
- tramite JSON

la lettura tramite url è in grado di recuperare la TD tramite quattro protocolli diversi:

- HTTP
- WebSocket
- CoAP
- MQTT

Interaction Affordances

Successivamente, dopo aver recuperato la TD, l'utente viene reindirizzato nella pagina in cui saranno mostrate le Interaction Affordances della TD letta e ha la possibilità di scegliere quale implementare; anche in questo caso non c'è un limite di scelta. Inoltre, l'utente ha la possibilità di scegliere se il Consumer debba essere anche una Thing, cioè gli viene data la possibilità di esporre Properties, Actions ed Events come avviene in fase di creazione della Thing vista precedentemente.

3.5.4 Modifica Consumer

Come descritto precedentemente nel processo di modifica della Thing, anche in questo caso è possibile modificare il Consumer sia tramite form HTML sia tramite editor.

3.5.5 Compilazione e upload dello sketch

Dopo che l'utente avrà creato la Thing o il Consumer, sarà per lui possibile compilare e fare l'upload dello sketch direttamente dall'interfaccia grafica; questa funzione chiamerà l'environment, che si occuperà di gestire tutti i passaggi appena elencati.

3.5.6 Feedback

Durante la fase di compilazione e upload vengono restituiti i feedback dei risultati in modo che l'utente abbia pieno controllo su ciò che accade durante il collegamento con la board, cosa che gli consente di gestire eventuali errori.

Il front-end comunica con il back-end e con l'environment attraverso API create per le operazioni appena descritte.

3.6 Back-end

Il back-end del software espone le API al front-end, riceve i dati ed esegue i vari compiti da svolgere in base al tipo di richiesta. Per la parte back-end sono state previste due componenti: Servient Builder e Microcontroller Engine. Il Servient Builder gestisce tutta la parte relativa agli sketch, come ad esempio creazione sketch, modifica, eliminazione ecc. Il Microcontroller Engine gestisce l'interazione con le board oltre che a compilare lo sketch e scaricare le librerie Arduino. Come Storage Services è stato usato MongoDB, mentre come Object Storage è stato usato MinIO.
Di seguito verranno dettagliate le due componenti:

3.6.1 Servient Builder

A livello implementativo il Servient Builder è costituito da due Controller: uno gestisce le richieste che riceve dal Client per la gestione della Thing e uno gestisce le richieste che riceve dal Client per la gestione del Consumer. In entrambi i casi espone gli endpoint HTTP per le operazioni di POST, PUT, GET, DELETE.
Quando riceve la richiesta POST per creare la Thing si comporta nel seguente modo:

Thing

- **Inizializzazione**
  Recupera lo sketch preconfigurato relativo alla creazione della Thing, crea una copia nominandolo col nome del progetto da creare e salva lo schema ricevuto nello Storage Services e nell'Object Storage lo sketch creato.

- **Lettura dati schema**
  Recupera i dati di build ricevuti e inizia a popolare lo sketch tramite

l'utilizzo della libreria fs (File System) che consente di interagire con il File System del Sistema Operativo.

- **Lettura Interaction Affordances**
  Legge le Properties, le Actions e gli Events presenti nello schema ricevuto e, in base ai Protocol Binding inserisce snippet di codice relativo al Protocol Binding nello sketch.

- **Creazione Thing Description**
  Infine, dopo aver ottenuto tutti i dati delle Interaction Affordances, crea la Thing Description, che verrà esposta dalla board sui Protocol Binding scelti dall'utente, e salva lo sketch nell'nello Storage Services.

Come accennato precedentemente, esistono due tipi di modifica: tramite form HTML e tramite editor. In entrambi i casi, per eseguire le modifiche, riceve richieste HTTP PUT ed esegue la seguente procedura:

- **Recupero schema form HTML**
  Recupera i dati dallo schema precedentemente salvato e i dati del nuovo schema ricevuto. Tali dati vengono salvati in opportune variabili e array.

- **Controllo dati**
  Dopo aver salvato i valori nelle variabili o negli array controlla se sono stati aggiunti, modificati o rimossi nuove Properties, Actions o Events.

- **Modifica sketch**
  Se ci sono Properties, Actions o Events da eliminare recupera le precedenti inserite nello sketch tramite espressioni regolari che restituiscono gli snippet di codice relativi alla ricerca effettuata, e li rimuove. Se ci sono nuove Properties, Actions o Events, aggiunge i relativi snippet di codice sullo sketch con i nuovi dati nelle opportune posizioni delineate tramite commenti.

- **Modifica Schema**
  La modifica dello schema, che compone il form HTML, viene aggiornato semplicemente salvando il nuovo schema ricevuto nello Storage Services sovrascrivendo il precedente.

- **Modifica Thing Description**
  Successivamente viene aggiornata la Thing Description con i nuovi dati e salvata nello sketch.

- **Salvataggio Sketch**
  Infine viene salvato lo sketch nell'Object Storage e nel File System.

La modifica tramite editor avviene nel seguente modo:

- **Lettura sketch**
  Viene recuperato e processato il nuovo sketch modificato dall'utente e tramite espressioni regolari vengono recuperate le Interaction Affordances; i dati ottenuti vengono salvati in opportune variabili o array;

- **Lettura schema**
  Viene recuperato lo schema precedentemente salvato nello Storage Services e lette Interaction Affordances, tali dati vengono poi salvate su opportune variabili o array;

- **Controllo dati**
  Vengono controllate le Interaction Affordances recuperate dalla lettura del codice e vengono confrontate con quelle lette nello schema salvato precedentemente;

- **Modifica schema**
  In base all'aggiunta di nuove Interaction Affordances o alla rimozione viene aggiornato lo schema e viene salvato nello Storage Services.

Le richieste GET recuperano lo schema o lo sketch e lo restituiscono al Client per mostrarlo all'utente, infine la richiesta DELETE elimina tutti i dati del progetto dallo Storage Services, dall'Object Storage e dal File System.

Consumer

Anche nel caso della gestione del Consumer le operazioni sono più o meno simili a quelle della gestione della Thing; di seguito verrà elencato il comportamento di tale gestione:

- **Inizializzazione**
  Recupera lo sketch preconfigurato relativo alla creazione del Consumer, crea una copia nominandolo col nome del progetto da creare e lo salva nell'Object Storage.

- **Lettura dati schema**
  Recupera i dati di build ricevuti e inizia a popolare lo sketch tramite l'utilizzo della libreria fs (File System) che consente di interagire con il File System del Sistema Operativo.

- **Lettura Interaction Affordances**
  Legge le Interaction Affordances di Properties, Actions ed Events abilitate dall'utente e le eventuali Properties, Actions ed Events create per essere anche una Thing, presenti nello schema ricevuto; in base ai Protocol Binding e al tipo di interazione (lettura o scrittura) inserisce gli snippet di codice relativi nello sketch.

- **Creazione Thing Description**
  Se le Properties, Actions ed Events sono state create anche per il Consumer viene creata la Thing Description che verrà esposta dalla board sui Protocol Binding scelti dall'utente.

- **Salvataggio sketch**
  Infine salva lo sketch nell'Object Storage.

Anche per il Consumer è possibile modificare il codice tramite editor e tramite form HTML. Quando viene ricevuta una richiesta HTTP PUT per la modifica tramite form HTML viene eseguita la seguente procedura:

- **Recupero schema form HTML**
  Recupera i dati dallo schema precedentemente salvato e i dati del nuovo schema ricevuto. Tali dati vengono salvati in opportune variabili e array.

- **Controllo dati**
  Dopo aver salvato i valori nelle variabili o negli array, controlla se sono state abilitate, modificate o rimosse Interaction Affordances dallo schema.

- **Modifica sketch**
  Se sono state abilitate Interaction Affordances dallo schema oppure se è stato cambiato il tipo di interazione (lettura o scrittura), recupera gli snippet di codice precedentemente aggiunti ed esegue le opportune modifiche. Se era stata creata anche la parte Thing sul Consumer controlla se ci sono Properties, Actions o Events da eliminare o da modificare, altrimenti, se ci sono nuove Properties, Actions o Events, aggiunge i relativi snippet di codice sullo sketch con i nuovi dati. Queste operazioni sullo sketch vengono effettuate tramite espressioni regolari.

- **Modifica Schema**
  La modifica dello schema, che compone il form HTML, viene aggiornato semplicemente salvando il nuovo schema ricevuto, quindi, sovrascrivendo il precedente;

- **Modifica Thing Description**
  Se sono presenti Properties, Actions o Events viene aggiornata la Thing Description con i nuovi dati e salvata nello sketch;

- **Salvataggio Sketch**
  Infine, viene salvato lo sketch nell'Object Storage e nel File System.

La modifica tramite editor avviene nel seguente modo:

- **Lettura sketch**
  Viene recuperato e processato il nuovo sketch modificato dall'utente e, tramite espressioni regolari, vengono recuperate le Interaction Affordances che vengono salvate in opportune variabili o array.

- **Lettura schema**
  Viene recuperato lo schema precedentemente salvato nello Storage Services e su di esso vengono recuperate le Interaction Affordances.

- **Controllo dati**
  Vengono controllate le Interaction Affordances recuperate dalla lettura del codice e vengono confrontate con quelle lette nello schema salvato precedentemente;

- **Modifica schema**
  Durante il processo di modifica del Consumer non è possibile aggiungere nuove Interaction Affordances in quanto le varie Interaction Affordances e i relativi Protocol Binding vengono rilevati durante la procedura di recupero della TD della Thing nel processo di creazione del Consumer. Quindi, le possibili operazioni di interazione con la Thing sono quelle rilevate nel processo iniziale di creazione del Consumer.

- **Salvataggio**
  Viene salvato il nuovo schema nello Storage Services e lo sketch nell'Object Storage.

Infine, le richieste GET recuperano lo schema o lo sketch e lo restituiscono al Client per mostrarlo all'utente, mentre la richiesta DELETE elimina tutti i dati del progetto dallo Storage Services, dall'Object Storage e dal File System.

3.6.2 Microcontroller Engine

Il Microcontroller Engine è il servizio che completa il funzionamento del software sviluppato e si occupa di interagire con i dispositivi embedded. La

funzione di questo servizio è l'installazione di nuove board, il recupero dello sketch, il download delle librerie presenti nello sketch, la compilazione dello sketch e l'upload dello sketch sulla board. Le informazioni dello sketch vengono rilevate tramite una query sullo Storage Services che, attraverso l'id del progetto, recupera tutte le informazioni. Per l'interazione vengono esposte quattro API: board-list, prepare, compile e upload e vengono forniti i feedback di ogni operazione, cosa che consente all'utente di capire e identificare gli errori che si verificano in tutte le fasi.

- **Prepare**
  Si occupa di recuperare le informazioni relative al progetto e di creare la cartella, se non esiste, in cui andrà inserito lo sketch; la cartella creata avrà il nome dello sketch.

- **Compile**
  Scarica le librerie necessarie al funzionamento dello sketch e compila lo sketch.

- **Upload**
  Si occupa di fare l'upload dello sketch sul dispositivo embedded.

- **Board-list**
  Mostra le porte disponibili sul sistema operativo.

Per ognuno di questi passaggi viene restituito uno dei due feedback in formato JSON:

- {"status": 200, "message": "informazioni relative alla prossima procedura"}

- {"status": 500, "message": "informazioni relative all'errore che ha causato l'interruzione"}

Questo servizio viene chiamato dall'utente tramite apposito bottone presente nell'interfaccia grafica.

Capitolo 4

Implementazione

In questo capitolo verrà illustrato il modo in cui ogni componente è stato sviluppato per raggiungere il suo obiettivo e le tecnologie utilizzate per la realizzazione dei componenti.
Il framework è disponibile al seguente link: https://github.com/UniBO-PRISMLab/micro-wot-servient-framework

4.1 Tecnologie

Tra le varie tecnologie attualmente presenti nel campo informatico sono state scelte le tecnologie che hanno una community ben consolidata e quelle tecnologie che hanno frequenti rilasci di nuove versioni, in questo modo il progetto può godere di costanti aggiornamenti nel tempo ed evitare di diventare un sistema legacy. Di seguito verranno elencate le tecnologie scelte.

4.1.1 Front-end

Angular

Angular è una piattaforma e un framework open source per la creazione di applicazioni client single page in HTML e TypeScript. Angular è scritto in TypeScript. Gli elementi costitutivi di base del framework Angular so-

no i Components Angular organizzati in NgModules. NgModules raccoglie il codice in set funzionali. Un'applicazione Angular è definita da un insieme di NgModules. Un'applicazione ha sempre almeno un modulo root che abilita il bootstrap. I Components definiscono le viste (view), che sono insiemi di elementi grafici che Angular può scegliere e modificare in base alla logica e ai dati del programma e utilizzano servizi (services) che forniscono funzionalità specifiche non direttamente correlate alle viste, cioè forniscono la logica del componente. I service providers possono essere iniettati nei Components come dipendenze, rendendo il codice modulare, riutilizzabile ed efficiente. I Modules, Components e Services sono classi che utilizzano decorators. I decorators forniscono metadati che indicano ad Angular come usarli. I metadati per una classe Components vengono associati ad un modello che definisce una vista. Un modello combina l'HTML con le direttive Angular e i binding markup che consentono ad Angular di modificare l'HTML prima di renderizzarlo per la visualizzazione. I metadati per una service class forniscono le informazioni di cui Angular ha bisogno per renderle disponibili ai Components tramite l'inserimento delle dipendenze (Dependency injection). I Components di un'applicazione in genere definiscono molte viste, disposte gerarchicamente. Angular fornisce il servizio Router per aiutare lo sviluppatore a definire i percorsi di navigazione tra le viste. Il router fornisce sofisticate capacità di navigazione all'interno del browser [1].

In questo progetto Angular è stato utilizzato per sviluppare la parte frontend dell'applicazione.

Librerie Front-end

Di seguito saranno elencate le librerie usate nel Front-end che completano l'interfaccia:

- **JSON editor**
  JSON editor, libreria Javascript open source, prende uno schema JSON e lo usa per generare un modulo HTML. Ha il pieno supporto per

JSON Schema versione 3 e 4 e può integrarsi con diversi framework CSS popolari (Bootstrap, Spectre, Tailwind) [15].
In questo progetto è stata usata per creare i form HTML nel Front-end in quanto restituiscono un oggetto JSON con tutti i dati dell'utente.

- **Monaco Editor**
  Monaco Editor è l'editor di codice utilizzato su Visual Studio Code sviluppato da Microsoft [19].
  In questo progetto è stato utilizzato per dare la possibilità all'utente di visualizzare e modificare il codice sorgente dello sketch sul Browser;

Interfaccia

Di seguito verrà descritta l'interfaccia grafica, realizzata attraverso Angular, mostrando il modo in cui l'utente interagisce con essa. L'utente, dopo aver inizializzato l'applicazione, navigherà attraverso il browser al seguente URL: localhost:4200/ in cui gli verrà mostrata l'interfaccia grafica del progetto, che costituisce il punto di ingresso dei dati e della gestione dell'interazione con le board.
La creazione della Thing è composta da un form HTML in cui l'utente può inserire le informazioni principali della Thing e le varie Interaction Affordances, un esempio è visibile in figura 4.3.
Dopo il salvataggio dei dati, l'utente vedrà il suo progetto nella lista dei progetti, in cui ha la possibilità di intraprendere le seguenti azioni:

- Modifica tramite editor;
- Modifica tramite form HTML;
- Upload sketch;
- Eliminazione progetto.

La modifica tramite form HTML si presenta uguale alla pagina di creazione della Thing, con la differenza che in questo caso i campi del form sono

già compilati; in questa sezione l'utente può modificare la Thing esistente aggiungendo, modificando o eliminando le Interaction Affordances. In figura 4.3 è possibile vedere la sezione di modifica tramite form HTML.

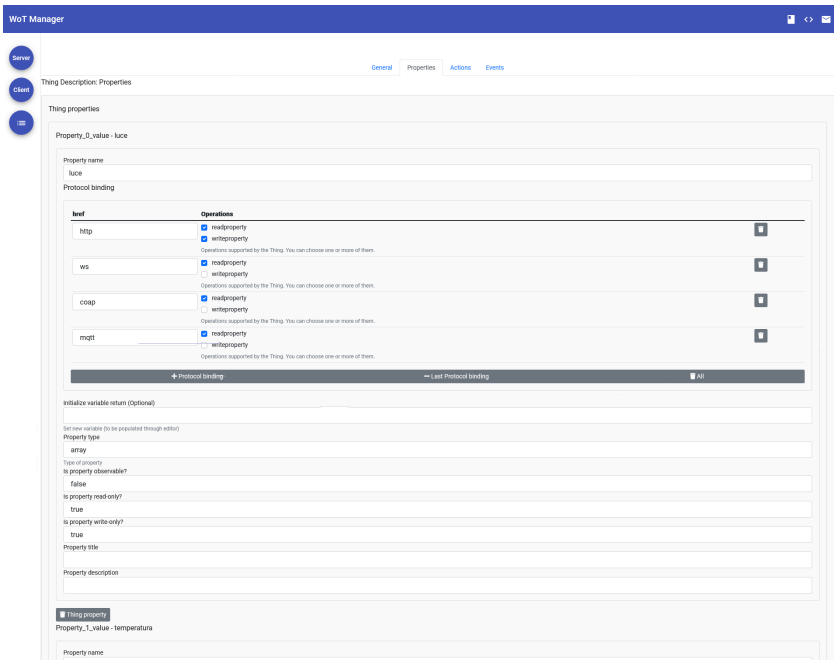


Figura 4.1: GUI: form editor Thing

La modifica tramite editor mostra un code editor sul browser in cui l'utente ha la piena libertà di modifica del codice sorgente. Questa sezione mostra lo sketch che girerà sulla board. In figura 4.4 è possibile vedere cosa l'utente vede nella sezione di modifica tramite editor.

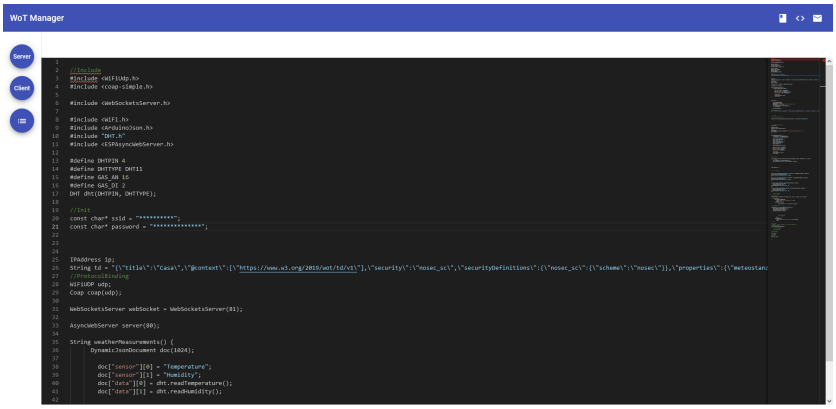


Figura 4.2: GUI: code editor Thing

L'utente ha la possibilità di compilare il codice e fare l'upload direttamente dall'interfaccia grafica cliccando sul bottone upload presente nella pagina della lista dei progetti. L'utente vedrà il processo di download delle librerie dello sketch, il processo di compilazione del codice e il processo di upload del codice. Per ogni passaggio appena descritto vengono mostrati i feedback di successo o fallimento in base all'esito dell'operazione e in caso di fallimento vengono mostrati i dettagli dell'errore.

Per quanto riguarda la creazione del Consumer, l'interfaccia si presenta in maniera diversa perché diverse sono le operazioni da fare per il Consumer. Nello specifico, se l'utente decide di creare un Consumer gli viene mostrata la pagina, visibile in figura 4.5, in cui ha due metodi di rilevazione della Thing Description della Thing: la prima è inserire l'URL e la seconda è inserire manualmente la Thing Description tramite editor.

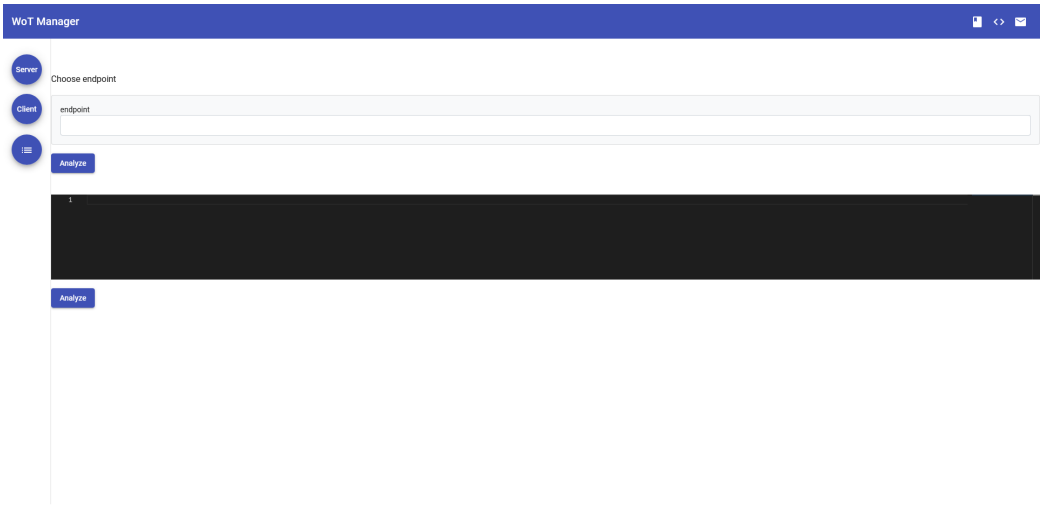


Figura 4.3: GUI: lettura Thing Description

Successivamente l'utente verrà reindirizzato in una pagina in cui gli verranno mostrate le Interaction Affordances presenti nella Thing Description rilevata, dove potrà scegliere quale Interaction Affordances abilitare in base ai Protocol Binding disponibili e scegliere in che modo interagire con esse. Inoltre, l'utente ha la possibilità di far diventare il Consumer anche una Thing abilitando Properties, Actions o Events. Infatti, come è possibile vedere in figura 4.6, l'utente, oltre alla sezione delle Interaction Affordances rilevate, ha anche a disposizione le sezioni relative alla creazione della Thing attraverso la compilazione delle Interaction Affordances: Properties, Actions ed Events.

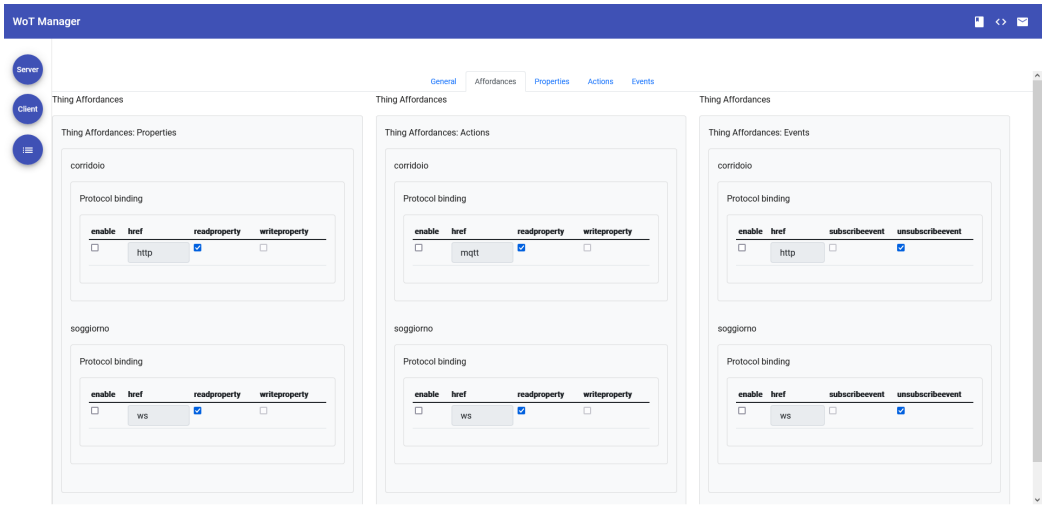


Figura 4.4: GUI: scelta Interaction Affordances Consumer

Anche in questo caso, dopo aver salvato i dati l'utente vedrà il Consumer creato nella pagina della lista dei progetti dove vengono mostrati sia i Consumer che le Thing. In questa pagina, l'utente, come mostrato in figura 4.4, ha la possibilità di intraprendere le seguenti azioni:

- Modifica tramite editor;
- Modifica tramite form HTML;
- Upload sketch;
- Eliminazione progetto.

La modifica tramite form HTML si presenta uguale alla pagina di creazione del Consumer, con la differenza che in questo caso i campi del form sono già compilati. In questa sezione l'utente può modificare il Consumer esistente aggiungendo, modificando o eliminando le Interaction Affordances.

La modifica tramite editor mostra un code editor sul browser in cui l'utente ha la piena libertà di modifica del codice sorgente. Questa sezione mostra lo sketch che girerà sulla board.

4.1.2 Back-end

NestJS

Nest (NestJS) fornisce un'architettura applicativa pronta all'uso che consente a sviluppatori e team di creare applicazioni altamente testabili, scalabili e di facile manutenzione. L'architettura è fortemente ispirata da Angular. Nest è un framework per la creazione di applicazioni lato server Node.js efficienti e scalabili. Utilizza JavaScript, supporta completamente TypeScript (ma consente ancora agli sviluppatori di programmare in JavaScript puro) e combina elementi di OOP (Programmazione orientata agli oggetti), FP (Programmazione funzionale) e FRP (Programmazione reattiva funzionale). Nest utilizza robusti framework HTTP Server come Express (impostazione predefinita) e opzionalmente può essere configurato per utilizzare anche Fastify. Nest fornisce un livello di astrazione superiore a framework Node.js comuni (Express/Fastify), ma espone anche le loro API direttamente allo sviluppatore. Ciò offre agli sviluppatori la libertà di utilizzare la miriade di moduli di terze parti disponibili per la piattaforma sottostante. I componenti architetturali di Nest sono i seguenti:

- **Controllers**
  I controllers sono responsabili della gestione delle richieste in arrivo e della restituzione delle risposte al client. Lo scopo di un controller è quello di ricevere richieste specifiche per l'applicazione. Il meccanismo di routing controlla quale controller riceve richieste. Spesso, ogni controller ha più di una route e routes diversi possono eseguire azioni diverse. Per creare un controller di base, vengono utilizzati classi e decoretors. I decoretors associano le classi ai metadati richiesti e consentono a Nest di creare una mappa di routing legando le richieste ai controller corrispondenti (figura 4.1).

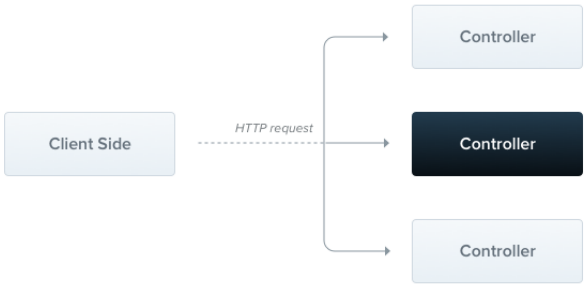


Figura 4.5: NestJS: richiesta client al controller [2]

- **Providers**
  I provider sono un concetto fondamentale in Nest. Molte delle classi Nest di base possono essere trattate come providers. L'idea principale di un provider è che può essere iniettato come dipendenza nei controller o in altri provider, ciò significa che gli oggetti possono creare varie relazioni tra loro.

- **Modules**
  Un modulo è una classe annotata con un decoratore @Module(). Il decoratore @Module() fornisce i metadati che Nest utilizza per organizzare la struttura dell'applicazione. Ogni applicazione ha almeno un modulo, il modulo root. Il modulo root è il punto di partenza utilizzato da Nest per creare il grafo applicativo, come è possibile vedere in figura 4.2) cioè la struttura dati interna che Nest utilizza per risolvere le relazioni e le dipendenze tra modulo e provider. Sebbene le applicazioni molto piccole possano teoricamente avere solo il modulo root, questo non è il caso tipico. I moduli sono fortemente raccomandati come un modo efficace per organizzare i components [2].

Nest all'interno del progetto è stato utilizzato per sviluppare il backend dell'applicazione.

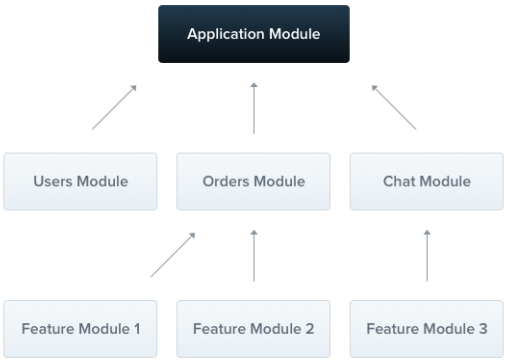


Figura 4.6: NestJS: application graph [2]

Arduino CLI

Arduino CLI (Command Line Interface) è la soluzione all-in-one open source per codificare, compilare e caricare codici sulle schede Arduino utilizzando il prompt dei comandi (Windows) e terminali (Linux e MAC). Oltre ad essere uno strumento autonomo, Arduino CLI è il cuore di tutti i software di sviluppo Arduino ufficiali (Arduino IDE, Arduino Web Editor) [3].

In questo progetto il Arduino CLI è utilizzato da un wrapper in Python, pyduinocli [26], una libreria che consente di utilizzare i comandi di Arduino CLI in Python. I comandi di Arduino vengono eseguiti dall'Environment, che si occupa di interagire con le board.

MongoDB

MongoDB è un DBMS non relazionale, orientato ai documenti. Classificato come un database di tipo NoSQL, MongoDB si allontana dalla struttura

tradizionale basata su tabelle dei database relazionali in favore di documenti in stile JSON con schema dinamico, rendendo l'integrazione di dati di alcuni tipi di applicazioni più facile e veloce [21].

MongoDB è lo Storage Services utilizzato in questo progetto e si occupa di salvare gli schemi dei form HTML compilati dall'utente.

MinIO

MinIO è un High Performance Object Storage rilasciato sotto GNU Affero General Public License v3.0. Può gestire dati non strutturati come foto, video, file di registro, backup e container image. Rispetto ai database distribuiti, che si occupano di metadati mutevoli e linguaggio di query, MinIO si occupa di gestire BLOB di dati non modificabili [20].

In questo progetto Minio è l'Object Storage utilizzato per salvare gli sketch delle Thing e dei Consumer.

Docker

Docker è una piattaforma aperta per lo sviluppo e l'esecuzione di applicazioni, consente di separare le applicazioni dall'infrastruttura in modo da poter distribuire rapidamente il software. Docker offre la possibilità di creare pacchetti ed eseguire applicazioni in un ambiente debolmente isolato chiamato container. Un container è un'unità software standard che racchiude il codice e tutte le sue dipendenze in modo che l'applicazione venga eseguita in modo rapido e affidabile da un ambiente di elaborazione all'altro. Un'immagine del contenitore Docker è un pacchetto software leggero, autonomo ed eseguibile che include tutto il necessario per eseguire un'applicazione: codice, runtime, strumenti di sistema, librerie di sistema e impostazioni. Le immagini dei container diventano container in fase di runtime e, nel caso dei container Docker, le immagini diventano container quando vengono eseguite su Docker Engine. Disponibile per applicazioni basate su Linux e Windows,

il software containerizzato funzionerà sempre allo stesso modo, indipendentemente dall'infrastruttura. I container isolano il software dal suo ambiente e assicurano che funzioni in modo uniforme nonostante le differenze, ad esempio tra sviluppo e staging. L'isolamento e la sicurezza consentono di eseguire più container contemporaneamente su un determinato host. I container sono leggeri e contengono tutto il necessario per eseguire l'applicazione, quindi non è necessario fare affidamento su ciò che è attualmente installato sull'host [13].

In questo progetto è stato usato docker per eseguire tutte le tecnologie usate nel progetto come container isolati che comunicano tra loro.

Librerie Back-end

Di seguito saranno elencate le librerie usate nel Back-end:

- **pyduinocli**
  Una libreria wrapper di arduino-cli per semplificare le chiamate ad arduino-cli da uno script python. Fornisce un modo chiaro e semplice per usare arduino attraverso un programma python [26].
  In questo progetto questa libreria consente di gestire tutta la parte relativa all'interazione con la board oltre che a scaricare tutte le librerie necessarie per lo sktech;

- **Flask**
  Flask è un framework per applicazioni Web WSGI (Web Server Gateway Interface) in Python. Inizialmente era un semplice wrapper di Werkzeug e Jinja ed è diventato uno dei framework di applicazioni Web Python più popolari [7].
  In questo progetto Flask è stato utilizzato per creare le API che eseguono specifiche funzioni di Arduino CLI attraverso la libreria pyduinocli nell'Environment.

Librerie Arduino

Le librerie Arduino utilizzate per l'implementazione dei Protocol Binding sono open source, tra le più utilizzate dagli sviluppatori e hanno frequenti rilasci di aggiornamento. Le librerie scelte sono le seguenti:

- EspMQTTClient [24]
- ESPAsyncWebServer [23]
- CoAP-simple-library [12]
- arduinoWebSockets [18]

Capitolo 5

Validazione

Nella parte di validazione sono stati effettuati tre esperimenti per verificare che dagli sketch generati dal framework non venga introdotto overhead nei protocolli di comunicazione. Il primo esperimento consiste nell'andare a misurare il valore di Round Trip Time (RTT) di ogni singolo protocollo eseguito nella Thing e nel Consumer. In questo esperimento vengono previste quattro diverse misurazioni: la prima effettuata utilizzando il protocollo HTTP, la seconda utilizzando il protocollo CoAP, la terza utilizzando WebSocket e infine la quarta utilizzando MQTT. Il secondo esperimento consiste nel misurare i valori di RTT di ogni singolo protocollo ma, a differenza del primo esperimento, in questo caso tutti e quattro i protocolli vengono eseguiti contemporaneamente sia nella Thing che nel Consumer con la sola condizione che il Consumer possa scegliere con quale protocollo comunicare in base al verificarsi di determinate condizioni. Ad esempio, se durante la comunicazione in HTTP il Consumer attende molto tempo per ricevere i dati dalla Thing, esso cambia automaticamente protocollo con uno di quelli abilitati. Il terzo esperimento consiste nel replicare i primi due con l'unica differenza che in questo caso viene effettuata un'implementazione manuale dello sketch. Quindi, per la validazione del progetto sono stati misurati i valori di RTT dei Protocol Binding HTTP, CoAP, WebSocket ed MQTT tramite la creazione di due scenari reali. I valori di RTT sono stati rilevati da due implemen-

tazioni di codice differenti: una creata tramite il framework sviluppato e l'altra implementata manualmente. I valori di RTT sono stati rilevati per capire se gli sketch generati tramite questo framework aggiungono overhead ai protocolli di comunicazione. Inoltre, è stata misurata la grandezza degli sketch utilizzati e la RAM utilizzata dalle due implementazioni differenti degli sketch.

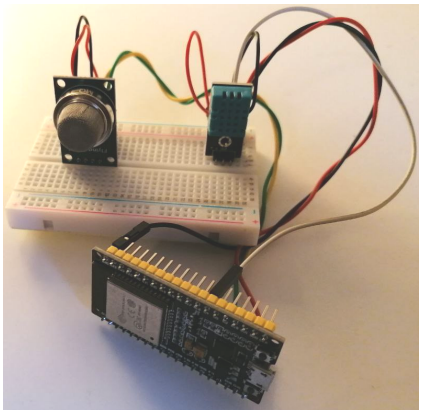
Per la creazione di tali scenari sono stati utilizzati i seguenti sensori, attuatori e dispositivi:

- **DHT11**: un sensore che rileva temperatura e umidità;
- **Active buzzer**: un buzzer che emette suoni al verificarsi di determinate condizioni;
- **Led**: due led, rosso e verde, che si accendono al verificarsi di determinate condizioni;
- **MH MQ sensor**: un sensore di rilevazione del gas;
- **ESP32**: due dispositivi programmabili.

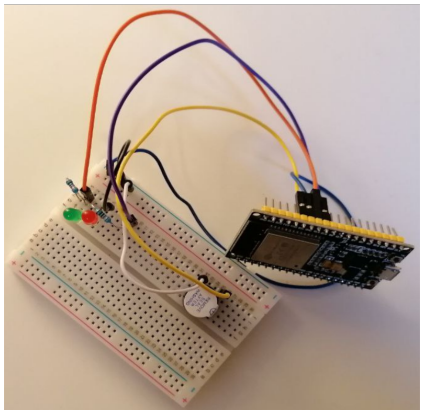
Per la rappresentazione di tali scenari il sistema creato è composto da due ESP32: uno ha il ruolo di Thing e l'altro ha il ruolo di Consumer. Come è possibile vedere in figura 5.1(a), la Thing è composta dal sensore di rilevazione del gas e dal sensore di rilevazione di temperatura e umidità. Per l'implementazione del Consumer, visibile in figura 5.1(b), sono stati inseriti due attuatori: il buzzer e i led, che si attivano al raggiungimento di determinate condizioni.

5.1 Valutazione delle performance

Come accennato precedentemente, per la valutazione dei dati sono stati creati due scenari reali in cui Thing e Consumer comunicano attraverso i Protocol Binding. Di seguito verranno descritti gli scenari creati:



(a) Validazione: Thing



(b) Validazione: Consumer

Figura 5.1: Sistema di validazione

5.1.1 Singoli protocolli

In questo test è stato creato uno scenario in cui vengono misurati i RTT dei vari Protocol Binding; per far ciò si è proceduto nel seguente modo:

Thing

Tramite il progetto sviluppato è stata creata una Thing con un Interaction Affordance di tipo Property che permette di leggere i dati rilevati dai sensori di temperatura, umidità e gas ed è stata esposta nella rete attraverso quattro protocolli: HTTP, CoAP, WebSocket ed MQTT. Successivamente, lo sketch prodotto è stato inserito all'interno del dispositivo ESP32 attraverso l'utility presente nel progetto. Tale Thing espone la sua Thing Description ed ha lo scopo di rendere disponibile i dati rilevati dai sensori.

Consumer

Tramite l'applicazione da me sviluppata è stata recuperata la Thing Description esposta dalla Thing precedentemente creata. Sulla base della TD sono stati abilitati i vari protocolli con cui il Consumer consumerà la Thing.

Dopo che la Thing e il Consumer sono stati creati, inizia la fase di sperimentazione. Il primo test consiste nella rilevazione dei dati di RTT dei singoli protocolli. Durante questa fase l'analisi viene fatta su tutti e quattro i protocolli disponibili singolarmente: HTTP, CoAP, WebSocket ed MQTT. La misurazione del RTT di ogni protocollo inizia la misurazione nel momento in cui viene richiesto il dato alla Thing e termina quando il dato richiesto viene restituito. Quindi, viene misurato il tempo che intercorre tra il momento della richiesta e la ricezione della risposta. Il payload è in formato JSON ed è di pochi byte. La durata della misurazione è di circa 20 minuti per protocollo.

5.1.2 Sistema adattivo

Il secondo test effettuato è il test adattivo, in cui vengono cambiati automaticamente i protocolli di comunicazione al verificarsi di determinate condizioni.
Per la creazione di tale scenario si è proceduto nel seguente modo:

Thing

Nella Thing vengono abilitati tutti i protocolli disponibili: HTTP, CoAP, WebSocket ed MQTT.

Consumer

Il Consumer creato per questo scenario si differenzia dal primo, descritto precedentemente, per l'abilitazione dei protocolli. In particolare, in questo contesto sono stati abilitati tutti i protocolli disponibili letti nella Thing Description della Thing e sono stati impostati dei parametri di attivazione dei singoli protocolli.

In questo scenario, il Consumer inizia la lettura dei valori dei sensori della Property utilizzando HTTP; ad un certo istante, dopo che si verifica una

condizione di ritardo nella lettura, il Consumer cambia protocollo leggendo la Property attraverso CoAP. Quindi, il sistema adattivo consente al Consumer di cambiare autonomamente protocollo di comunicazione ogniqualvolta si verifica un ritardo sul protocollo in uso. I valori misurati di RTT sono stati rilevati su tutti i protocolli utilizzati e sono stati salvati su InfluxDB [14].

5.1.3 Implementazione manuale

Dopo aver utilizzato il progetto per creare le Thing e i Consumer, in questo contesto è stata fatta un'implementazione manuale degli scenari creati precedentemente. Tale implementazione è stata fatta per valutare le differenze delle prestazioni tra ciò che è stato prodotto dal progetto e ciò che un utente crea senza l'utilizzo di tale progetto, misurandone, anche in questo caso, i valori di RTT sia per lo scenario adattivo sia per lo scenario in cui vengono misurate le prestazioni dei singoli protocolli.

5.2 Risultati

Il primo test effettuato misura il RTT dei singoli protocolli HTTP, CoAP, WebSocket ed MQTT per capire quali siano le differenze che questi protocolli hanno nello scenario IoT e per confrontarli con i risultati ottenuti dallo stesso scenario implementato manualmente. Il sistema che ha prodotto questi dati, visibili in figura 5.2, è stato creato tramite il framework. Come si può notare nel grafico in figura 5.2, è chiara la differenza di RTT che c'è tra i vari protocolli, in particolare HTTP ha un RTT di circa 800ms mentre gli altri protocolli hanno un RTT di gran lunga più basso. Per ogni protocollo sono stati effettuati circa 1000 misurazioni.

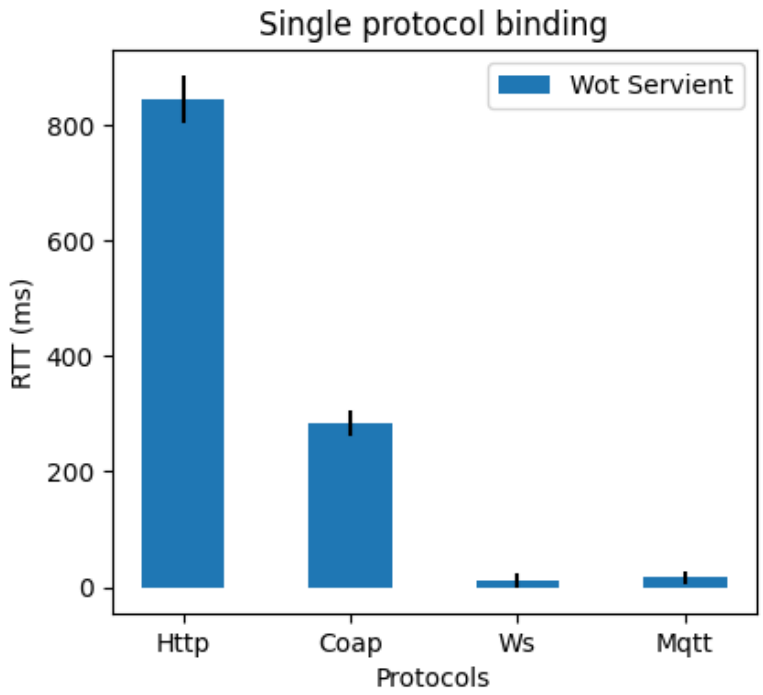


Figura 5.2: Valutazione performance: singol protocol (implementazione tramite progetto)

Nella figura 5.3, viene rappresentato il grafico che riporta il valore del RTT dei singoli protocolli HTTP, CoAP, WebSocket ed MQTT messi a confronto con i dati del grafico in figura 5.2. A differenza dei precedenti, questi valori sono stati rilevati dal sistema sviluppato manualmente tramite Arduino IDE. Dai dati visibili in figura 5.3 si può notare che il valore di RTT più alto è dato da HTTP e tale valore scende sotto i 100ms con l'utilizzo dei protocolli WebSocket ed MQTT. Inoltre, i valori di RTT ottenuti con l'implementazione manuale del sistema non si differenziano molto dai valori ottenuti dall'implementazione del sistema tramite framework. Infine, si può notare che il codice prodotto dal framework realizzato da questo lavoro di tesi non aggiunge overhead al sistema implementato col codice scritto manualmente, quindi è possibile utilizzare questo framework tenendo in considerazione tutti i limiti e i vantaggi esistenti dei protocolli utilizzati.

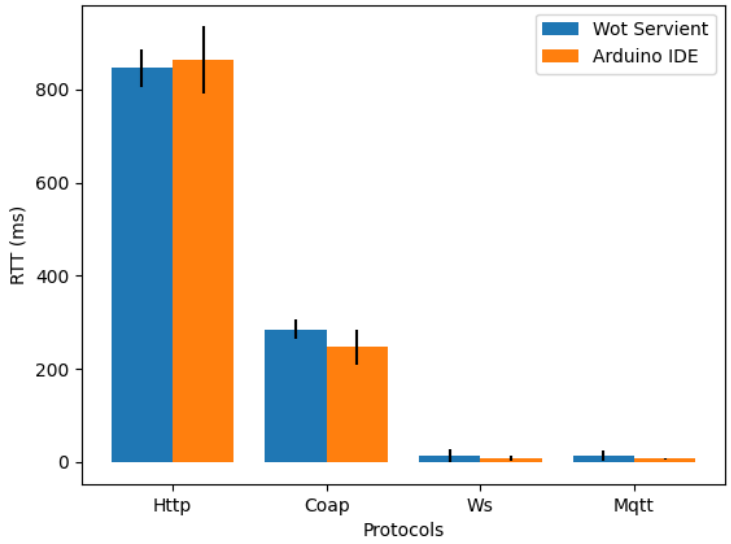


Figura 5.3: Valutazione performance: comparazione singol protocol (implementazione tramite progetto e manuale)

Il secondo test effettuato misura il valore del RTT dei singoli protocolli HTTP, CoAP, WebSocket ed MQTT ma, a differenza del precedente, ciò viene eseguito in un contesto adattivo nel quale il dispositivo ha la possibilità di cambiare protocollo di comunicazione al verificarsi di determinate condizioni. Come è possibile vedere in figura 5.4, i risultati prodotti da questo test non mostrano differenze con i risultati ottenuti dal test precedente in cui la lettura della Property avveniva con singoli protocolli per dispositivo. La figura 5.4 mostra i valori di RTT di ogni protocollo utilizzato dal sistema realizzato tramite il framework e il sistema realizzato manualmente. Nel grafico 5.4 è rappresentata anche la sequenza in cui questi protocolli vengono attivati nell'arco temporale in cui è stato eseguito il test. Dal grafico 5.4 si può notare che il RTT per la tecnologia HTTP, in entrambe le implementazioni, ha valori più alti 700ms, mentre scende sotto questa soglia con l'utilizzo del protocollo CoAP. Infine, si può notare che quando il sistema utilizza WebSocket o MQTT questi valori non superano i 100ms. Dai dati ottenuti è

possibile vedere che non ci sono particolari differenze tra l'implementazione dello sketch tramite framework e l'implementazione dello sketch manuale. Quindi, l'utilizzo di questo framework non comporta nessun ulteriore ritardo ai protocolli di comunicazione e, di conseguenza, è possibile utilizzarlo anche in un contesto adattivo tenendo conto dei limiti e dei vantaggi che questi protocolli hanno nel contesto IoT.

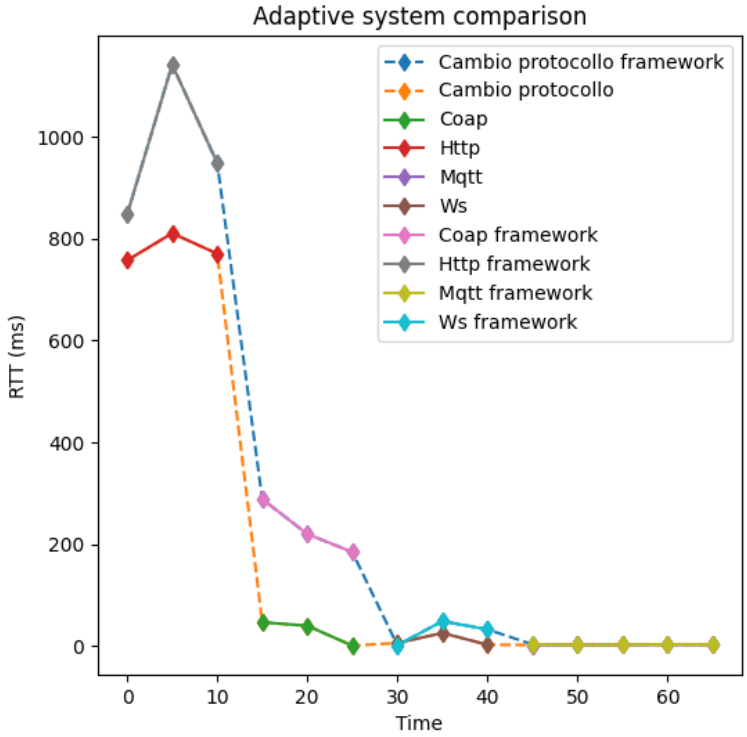


Figura 5.4: comparazione sistema adattivo

Infine, dai dati rilevati in fase di upload dello sketch si può affermare che lo sketch generato dal progetto occupa più o meno il 10% in più in termini di dimensione dello sketch rispetto a quello creato manualmente. Questo perché per rendere lo sketch generato dal framework leggibile all'utente sono stati inseriti variabili, funzioni e commenti che aiutano a comprendere le

singole parti del codice. Anche l'utilizzo della RAM non è superiore di quella utilizzata da uno sketch programmato manualmente.

Capitolo 6

Conclusioni e sviluppi futuri

In questo elaborato di tesi si è visto come è possibile affrontare il problema dell'utilizzo del paradigma W3C WoT nell'ambito dei microcontrollori. Tale lavoro si è reso necessario in quanto l'implementazione del paradigma W3C WoT ad oggi non viene utilizzata su sistemi di basso livello e l'unico progetto ufficiale che consente di implementare il paradigma W3C WoT è rappresentato da node-wot [10]. Node-wot è un'applicazione Javascript che opera su sistemi di alto livello, motivo per cui non è possibile vedere una sua implementazione in sistemi di basso livello quali sono ad esempio i microcontrollori. Per le motivazioni appena elencate, si è voluto realizzare un framework multipiattaforma che consente ad utenti e sviluppatori di progettare, compilare ed installare applicazioni WoT su dispositivi embedded implementando il paradigma W3C WoT. Tale framework è in grado di realizzare sketch per la creazione di Thing e Consumer dando la possibilità a chi ne usufruisce di creare la propria applicazione senza preoccuparsi dei requisiti proposti dallo standard W3C WoT e consentendo di avere pieno controllo sullo sketch creato grazie alle due possibili modifiche: modifica tramite editor e modifica tramite form HTML.
La realizzazione del presente progetto si basa sullo studio di una applicazione già esistente [25], alla quale sono stati aggiunti nuovi requisiti. Nello specifico, si è partiti dallo studio dell'applicazione esistente creando una lista di

funzionalità mancanti e bug riscontrati in fase di utilizzo. Successivamente è stato redatto un documento con i requisiti da soddisfare e gli obiettivi da raggiungere, tenendo conto di ciò che lo standard W3C WoT propone. Infine, è stato fatto uno studio sulle tecnologie da utilizzare per la realizzazione di questo lavoro. Si è scelto di utilizzare nuove tecnologie che hanno una community ben consolidata e un rilascio di aggiornamenti frequenti per consentire al progetto di aggiornarsi frequentemente, di evolversi con nuove funzionalità e di essere manutenibile.

Lo sviluppo del progetto si è rivelato molto complesso e costoso in termini di tempo in quanto le parti che lo compongono sono completamente diverse rispetto al progetto esistente. Di fatto è stata ripensata tutta l'architettura e sono stati utilizzati framework ben strutturati che integrano pattern architetturali già consolidati che facilitano la lettura del codice, consentendo al progetto di essere leggibile e manutenibile in futuro.

Un punto debole di questo framework è rappresentato dal fatto che non vi è la possibilità per l'utente o lo sviluppatore di decidere quali librerie utilizzare per implementare i protocolli di comunicazione.

Possibili sviluppi futuri vengono identificati in diverse funzionalità mancanti; tra queste le più importanti, che renderebbero il progetto usufruibile da una platea più ampia, sono il supporto a nuove board e l'aggiunta di nuovi Protocol Binding oltre ad HTTP, CoAP, WebSocket ed MQTT. Un altro possibile sviluppo futuro è l'aggiunta di nuovi protocolli di comunicazione nel Front-end, in particolare nella sezione di lettura della Thing Description, in quanto attualmente questo è possibile solo tramite HTTP, CoAP, WebSocket ed MQTT. Infine, il progetto potrebbe essere arricchito anche dall'inserimento di un monitor seriale che consente all'utente o allo sviluppatore di poter visualizzare eventuali informazioni in fase di sviluppo e dalla possibilità di aggiornare lo sketch senza collegare fisicamente la board al computer, per esempio tramite la funzionalità Over The Air (OTA).

Bibliografia

[1] URL: https://angular.io/.

[2] *A progressive node.js framework*. URL: https://nestjs.com/.

[3] Arduino. *Arduino/Arduino-CLI: Arduino Command Line Tool*. URL: https://github.com/arduino/arduino-cli.

[4] Kevin Ashton. *That 'internet of things' thing*. 2020. URL: https://www.rfidjournal.com/that-internet-of-things-thing.

[5] *Auto-ID*. URL: https://www.digital4.biz/whitepapers/auto-id-come-funziona-la-tracciabilita-e-perche-e-lunita-fondamentale-della-iot/.

[6] Wayne Beaton. *Eclipse Thingweb™*. 2022. URL: https://projects.eclipse.org/projects/iot.thingweb.

[7] *Flask*. URL: https://palletsprojects.com/p/flask/.

[8] Ala Al-Fuqaha et al. «Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications». In: *IEEE Communications Surveys Tutorials* 17.4 (2015), pp. 2347–2376. DOI: 10.1109/COMST.2015.2444095.

[9] Dominique Guinard. «A web of things application architecture: Integrating the real-world into the web». Tesi di dott. ETH Zurich, 2011.

[10] Dominique D. Guinard e Vlad M. Trifa. *Using the Web to Build the IoT*.

[11] Siham Al Hinai e Ajay Vikram Singh. «Internet of things: Architecture, security challenges and solutions». In: *2017 International Conference on Infocom Technologies and Unmanned Systems (Trends and Future Directions) (ICTUS)*. 2017, pp. 1–4. DOI: 10.1109/ICTUS.2017.8286004.

[12] Hirotakaster. *Hirotakaster/CoAP-simple-library*. URL: https://github.com/hirotakaster/CoAP-simple-library.

[13] *Home*. 2022. URL: https://www.docker.com/.

[14] *InfluxDB: Open source time series database*. 2022. URL: https://www.influxdata.com/.

[15] Json-Editor. *JSON-editor/JSON-editor: JSON schema based editor*. URL: https://github.com/json-editor/json-editor.

[16] Rafiullah Khan et al. «Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges». In: *2012 10th International Conference on Frontiers of Information Technology*. 2012, pp. 257–260. DOI: 10.1109/FIT.2012.53.

[17] Michael Lagally e Michael McCool. «IoT Interoperability with W3C Web of Things». In: *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC)*. 2022, pp. 1–5. DOI: 10.1109/CCNC49033.2022.9700546.

[18] Links2004. *LINKS2004/arduinowebsockets: Arduinowebsockets*. URL: https://github.com/Links2004/arduinoWebSockets.

[19] Microsoft. *Microsoft/Monaco-editor: A browser based code editor*. URL: https://github.com/microsoft/monaco-editor.

[20] Inc. MinIO. *High performance, kubernetes native object storage*. URL: https://min.io/.

[21] *MongoDB*. Gen. 2022. URL: https://it.wikipedia.org/wiki/MongoDB.

[22] Federico Montori et al. «Machine-to-machine wireless communication technologies for the Internet of Things: Taxonomy, comparison and open issues». In: *Pervasive and Mobile Computing* 50 (2018), pp. 56–81. ISSN: 1574-1192. DOI: https://doi.org/10.1016/j.pmcj.2018.08.002. URL: https://www.sciencedirect.com/science/article/pii/S1574119217303668.

[23] Me-No-Dev. *Me-no-dev/espasyncwebserver: Async web server for ESP8266 and ESP32*. URL: https://github.com/me-no-dev/ESPAsyncWebServer.

[24] plapointe6. *Plapointe6/Espmqttclient: WIFI and MQTT handling for ESP8266 and ESP32*. URL: https://github.com/plapointe6/EspMQTTClient.

[25] Federico Rachelli. «Micro-WoTServient: progettazione ed implementazione di uno stack software Web of Things per microcontrollori». Tesi di dott. URL: http://amslaurea.unibo.it/21606/.

[26] Renaud11232. *Renaud11232/pyduinocli: Pyduinocli is a wrapper library around Arduino-CLI to make the Arduino-cli calls easy from a python script.* URL: https://github.com/Renaud11232/pyduinocli.

[27] Sahrish Khan Tayyaba et al. «Software Defined Network (SDN) Based Internet of Things (IoT): A Road Ahead». In: *Proceedings of the International Conference on Future Networks and Distributed Systems* (2017).

[28] Lionel Sujay Vailshery. *IOT connected devices worldwide 2019-2030*. 2022. URL: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[29] *Web of things (wot) architecture*. URL: https://www.w3.org/TR/wot-architecture/.

[30] Erik Wilde. «Putting things to REST». In: (2007).

[31] *World Wide Web Consortium*. 2022. URL: https://it.wikipedia.org/wiki/World_Wide_Web_Consortium.

[32] Deze Zeng, Song Guo e Zixue Cheng. ≪The web of things: A survey≫. In: *Journal of Communications* 6 (gen. 2011), pp. 424–438.

Ringraziamenti

Ringrazio il mio relatore Professore Marco Di Felice e il mio correlatore Dottor Luca Sciullo per la loro disponibilità e per avermi permesso di realizzare questo progetto di tesi.
Ringrazio la mia famiglia, per avermi permesso di arrivare fin qui e avermi dato tanto supporto durante questo percorso.
Ringrazio Paola, per essermi sempre vicina anche nei momenti più difficili.
Ringrazio i miei amici di sempre e i nuovi conosciuti, per avermi sempre sostenuto durante questo percorso universitario.