Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea Magistrale in Informatica

Progettazione e sviluppo di un sistema di gestione dati IoT per applicazioni di monitoraggio strutturale

Relatore:
Chiar.mo Prof.
Marco Di Felice

Correlatore:
Dott.
Lorenzo Gigli

Presentata da:
Matteo Marchesini

Sessione IV
Anno Accademico 2018/2019

Sommario

Partendo da un'analisi dell'Internet of Things (IoT) e del Web of Things (WoT), degli scenari applicativi e delle criticità, viene proposta la realizzazione di un sistema destinato alla gestione di dati provenienti dal mondo IoT per il monitoraggio strutturale, finalizzato ad una manutenzione intelligente di impianti e opere civili. L'applicazione fornisce all'utente uno strumento con cui poter visualizzare e manipolare i dati emessi dalle reti di sensori IoT posizionati sulle strutture monitorate. Fornisce funzionalità di ricerca e filtraggio sui dati, approcciando ad un'architettura WoT secondo gli standard W3C. Il lavoro svolto rappresenta un componente dell'intera infrastruttura WoT proposta dal progetto di ricerca Mac4Pro.

Indice

Elenco delle figure

Elenco listati di codice

Capitolo 1

Introduzione

Nonostante il concetto di Internet of Things (IoT) non sia molto recente, negli ultimi anni si è assistito ad una crescita esponenziale del settore, che non si fermerà di certo negli anni a venire. Infatti se nel 2018 il numero di dispositivi connessi era di 23.14 miliardi ad oggi la cifra è di 30.73, e la stima per il 2025 è di raggiungere quota 75.44, con un mercato attuale che si aggira sui 250 miliardi di dollari spesi da utenti finali in tutto il mondo per soluzioni IoT [5][3].
Una delle applicazioni dell'IoT in forte crescita è il monitoraggio strutturale di infrastrutture ed opere pubbliche al fine di controllare e valutare lo stato di conservazione. L'attenzione sul tema è cresciuta soprattutto negli ultimi anni almeno in Italia, a seguito del crollo del Ponte Morandi che ha riportato in primo piano l'esigenza di cambiare radicalmente l'approccio alla manutenzione delle infrastrutture del nostro Paese. Sebbene l'IoT apporti benefici e vantaggi in termini di automazione, i numerevoli dispositivi presenti sul mercato comportano una criticità in termini di interoperabilità, poichè utilizzando protocolli e standard differenti rendono complessa la comunicazione tra dispositivi eterogenei. Al giorno d'oggi si è cercato di risolvere questo problema con il Web of Things (WoT), utilizzando gli standard e le tecnologie web già ampiamente diffusi e solidi, adatti a costruire un'infrastruttura altamente scalabile. La standardizzazione del WoT è iniziata nel 2015 per

merito del World Wide Web Consortium (W3C) il cui obiettivo è quello di ovviare alla frammentazione presente nel mondo IoT introducendo una serie di componenti che permettono di integrare piattaforme e domini applicativi diversi tra loro. Lo scopo del lavoro oggetto di questa tesi è la progettazione e implementazione di un sistema che permetta di gestire e visualizzare i dati IoT prodotti in ambito di monitoraggio strutturale, ovvero da sensori localizzati su strutture di vario genere, come ponti e circuiti idraulici. Infatti il progetto fa parte di un lavoro di ricerca a cui aderisce il DISI chiamato Mac4Pro, il cui obiettivo è fornire un'architettura per il monitoraggio strutturale. Quindi la piattaforma si inserisce all'interno di tale architettura, strutturata secondo i principi del WoT. Una peculiarità di questa piattaforma è la possibilità di manipolare e monitorare i dati in real-time, attraverso un'interfaccia intuitiva che permette l'utilizzo a qualsiasi tipologia di utente, purchè abbia una minima conoscenza della struttura monitorata e della rete sensoristica in essa presente.

In questo elaborato sono descritte le varie fasi di studio e di lavoro svolto per ottenere la realizzazione dell'applicazione. Nella prima parte (capitolo 2, 3, 4) viene presentato lo stato dell'arte. Il capitolo 2 presenta una panoramica del mondo IoT, con riferimento alla sua evoluzione e alle caratteritiche attuali. Quindi prende in esame il WoT descrivendone l'utilità e l'utilizzo, infine introduce il W3C e la sua composizione. Nel capitolo 3 viene analizzata l'architettura WoT, evidenziandone i casi d'uso, gli elementi principali e l'implementazione di una Servient. Il capitolo 4 fornisce una confronto degli strumenti di visualizzazione dati IoT attualmente presenti sul mercato, analizzandone le proprietà. La seconda parte dell'elaborato (capitolo 5, 6, 7) riguarda in maniera specifica l'applicazione sviluppata. Nel capitolo 5 viene presentata l'idea iniziale da cui è nato il progetto, nonchè i requisiti funzionali e tecnici e l'intera architettura. Passando al capitolo 6 vengono esposte le tecnologie utilizzate e i moduli implementati. Infine nel capitolo 7 viene mostrato lo scenario applicativo del progetto e i risultati ottenuti.

Parte I

Stato dell'arte

Capitolo 2

Panoramica generale

2.1 Internet of Things

Internet of Things (IoT) è un neologismo nato dall'esigenza di dare un nome al mondo degli oggetti reali connessi ad Internet. L'origine del termine risale al 1999 ed è stato introdotto per la prima volta da Kevin Ashton, ricercatore presso il MIT (Massachusetts Institute of Technology), per descrivere un sistema in cui Internet è connesso al mondo fisico tramite sensori onnipresenti.

Quindi possiamo notare che sebbene la diffusione della terminologia "Internet of Things" sia aumentata notevolmente solo in anni recenti, tale concetto ha origini ben remote e il suo significato si è raffinato ed evoluto nel corso degli anni a causa della convergenza delle tecnologie, arrivando alla complessità ed estensione che oggi conosciamo e ricoprendo un'ampia gamma di applicazioni come sanità, servizi e trasporti.

Esprimere la realtà dell'Internet of Things attraverso una sola definizione sarebbe riduttivo, tuttavia è possibile affermare che l'IoT può essere visto come un evoluzione radicale dell'Internet attuale che ne estende la potenza attraverso una rete di oggetti connessi che non solo raccoglie informazioni

dall'ambiente e permette di interagire con il mondo fisico, ma utilizza anche gli standard Internet esistenti al fine di fornire servizi, quali trasferimento di informazioni, analisi, applicazioni e comunicazioni.

A global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies. [6]

L'evoluzione del mondo IoT si compone di diverse fasi come illustrato nella figura 2.1. Nella fase primordiale l'IoT ha visto l'uso della tecnologia RFID (Radio Frequency Identification), con un utilizzo maggiore nella logistica, produzione farmaceutica, vendita al dettaglio ecc. L'emergere delle tecnologie sensoristiche wireless ha ampliato le capacità sensoriali dei dispositivi estendendo così il concetto di IoT all'intelligenza ambientale ed al controllo autonomo.

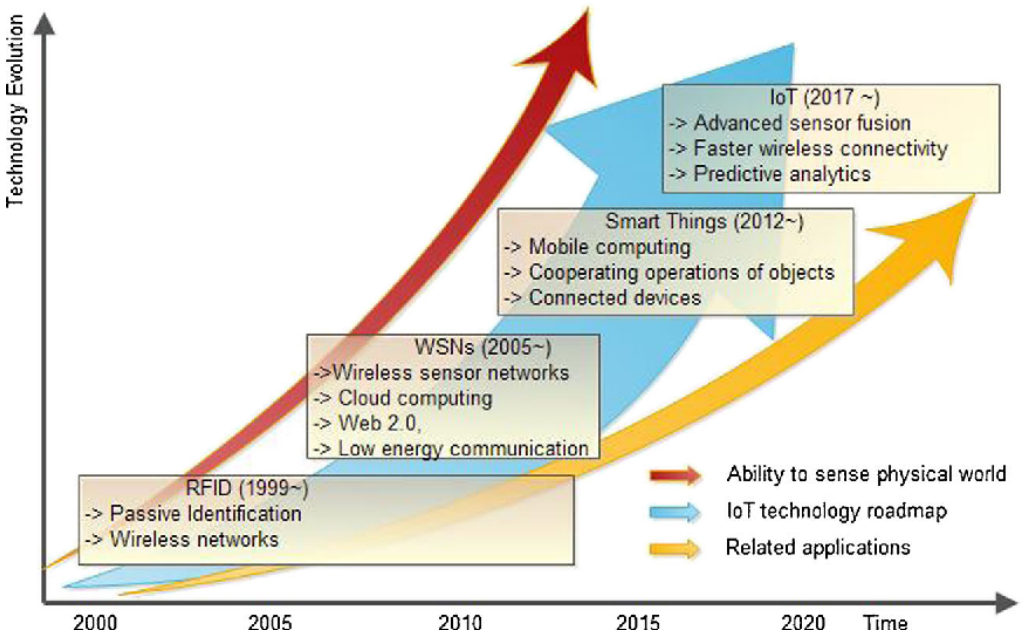

Figura 2.1: Evoluzione dell'IoT

Fonte: [10]

L'IoT comprende un vasto range di tecnologie, come le reti di sensori wireless (WSNs), RFID, NFC, cloud computing, comunicazioni wireless low-energy, sensori intelligenti (sensori in grado di compiere determinate azioni alla ricezione di determinati input), e molte altre [10]. Con il colossale progresso delle tecnologie e in particolare dei dispositivi embedded, il mercato ha visto la nascita di una nuova tipologia di oggetti: le Smart Things. Una Smart Thing (a cui da ora in poi faremo riferimento come Thing) è un oggetto fisico potenziato digitalmente con una delle seguenti caratteristiche:

- Sensori (temperatura, luce, movimento, ecc.)
- Attuatori (display, suono, motori, ecc.)
- Calcolo (logica ed esecuzione di programmi)
- Interfacce di comunicazione (cablate o wireless)

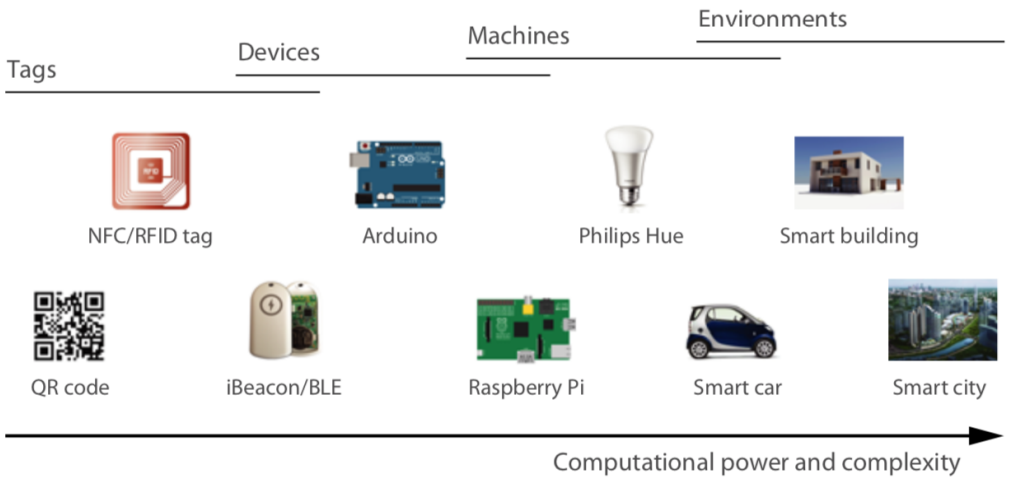

Figura 2.2: Esempi di Smart Things

Fonte:[15]

Le Thing estendono il mondo in cui viviamo abilitando un'ampia gamma completamente nuova di applicazioni (Figura 2.2). Integrando nuovi dispositivi nell'ambiente in cui viviamo, con il passare degli anni sarà possibile monitorare ed interagire con il mondo fisico con una risoluzione spaziale e temporale maggiore, riuscendo così a manipolare e controllare anche da remoto qualsiasi tipo di oggetto.

Concretamente, le thing nell'IoT possono variare da semplici prodotti muniti di tag Auto-ID (metodi di identificazione automatica come codici a barre, codici QR e tag NFC e RFID) che possono essere monitorati e rintracciati durante le spedizioni (qualsiasi oggetto comprato su Amazon dispone di tale tag) fino a sistemi più complessi come ad esempio una catena di montaggio in fabbrica, un edificio o addirittura una città.
Nell' IoT ciò che accomuna tutte le Thing è la prima parte del termine: Internet. Infatti ciò significa che una qualsiasi Thing (o almeno i suoi servizi o dati) è accessibile attraverso l'infrastruttura Internet esistente. Però ciò non implica che la Thing stessa debba essere direttamente connessa a Internet. La rete di comunicazione utilizzata può essere un metodo Auto-ID, una comunicazione radio a corto raggio (Bluetooth, ZigBee e simili) o la rete Wi-Fi all'interno di un edificio. Come appena accennato, il dominio IoT è suddiviso tra portocolli di rete low-energy (ZigBee, ZWave e Bluetooth), protocolli di rete tradizionali (Ethernet e WiFi) e connessioni cablate.

Questo costituisce il principale problema dell'Internet of Things, infatti oggi è praticamente impossibile costruire un ecosistema unico e globale di Thing che comunicano tra loro, in quanto viene a mancare l'esistenza di un protocollo applicativo universale per l'IoT che può funzionare tra le varie interfacce di rete disponibili. Ciò è dovuto anche al fatto che le imprese tecnologiche produttrici di tali dispositivi utilizzano piattaforma diverse l'una dall'altra rendendo così impossibile l'interazione tra dispositivi diversi.

Quindi, affinchè l'Internet of Things diventi qualcosa di reale, è necessario un unico protocollo universale a livello applicativo che permetta a dispositivi diversi e applicazioni di comunicare tra loro, senza tener conto della connessione fisica. È proprio da questo concetto che nasce il Web of Things (WoT): piuttosto che inventare un nuovo protocollo da zero, possiamo riutilizzare i protocolli, gli standard e schemi del Web che sono ampiamente diffusi e affidabili, con i quali è possibile costruire applicazioni altamente scalabili. In questo modo, intruducendo un'astrazione a livello applicativo, si rendono disponibili dati e servizi ad uno spettro più ampio di sviluppatori.

2.1.1 Monitoraggio strutturale

Negli ultimi anni un nuovo scenario di applicazione dell'IoT è il monitoraggio strutturale (Structural Health Monitoring) e i sistemi sviluppati in questo campo sono stati sperimentati con buoni risultati [9]. Tali sistemi sono principalmente dedicati al monitoraggio, ad esempio, di più parti di una struttura civile / industriale, con lo scopo di eseguire una manutenzione predittiva al fine di ridurre i costi di manutenzione ed aumentare la sicurezza dell'uomo.

Il monitoraggio strutturale mira a fornire, durante la vita di un'opera di ingegneria civile, informazioni sulla qualità delle materie prime e sul funzionamento di una singola parte o dell'intera struttura. Lo stato della struttura deve rimanere nel dominio di sicurezza specificato dall'ingegnere durante la progettazione, però questo stato può essere modificato da diversi fattori, quali il normale invecchiamento delle materie prime a causa dell'uso o del degrado chimico, l'azione dell'ambiente o eventi accidentali.

Proprio per questo SHM mira a identificare, rilevare e caratterizzare il degrado e il danno di tutti i tipi di strutture ingegneristiche. Il sistema SHM utilizza sensori per monitorare diverse quantità fisiche come accelerazione, trazione e sollecitazione di compressione, temperatura, umidità e così via.

Inoltre vengono utilizzati in genere anche diversi metodi non invasivi basati sulla distribuzione del sensore i punti specifici di controllo (stabiliti dagli esperti) e sulla fusione tra le informazioni sperimentali acquisite con i sensori e modelli matematici.

Grazie allo sviluppo del paradigma IoT il monitoraggio strutturale ha visto un miglioramento in termini di affidabilità ed efficienza rispetto ai progetti esistenti nei diversi scenari applicativi. Infatti con l'introduzione delle reti wireless vi è la possibilità di allocare facilmente sensori wireless nello spazio, aumentando il volume della struttura monitorata e riducendo i costi hardware rispetto all'adozione di una tecnologia di rete cablata ad hoc. I sensori utilizzati per SHM sono caratterizzati da capacità computazionali e comunicative, che li rendono appunto Thing. Tali reti di thing consentono di raccogliere informazioni dall'ambiente e permettono di eseguire misurazioni sincronizzate attraverso l'implementazione di opportuni algoritmi.

Dal punto di vista architetturale, un sistema di monitoraggio strutturale IoT si compone di 4 principali componenti, come visibile in figura 2.3:

- Thing sensori
- Gateway
- Remote control e service room (RCSR)
- Open platform communication server (OPC)

La prima parte del sistema è composta dalla rete di sensori che adibiscono alla funzione di monitoraggio dell'integrità strutturale, definita in base al tipo di fenomeno fisico (correlato al danno). Tali sensori vengono selezionati in base al tipo di struttura e al fenomeno fisico da rilevare. Ognuno di essi produce un segnale che viene inviato al sottositema adibito all'acquisizione e alla memorizzazione.
In genere questi sensori sono strutturati come un nodo e sono in grado di

misurare ad esempio temperatura o vibrazioni della struttura monitorata.

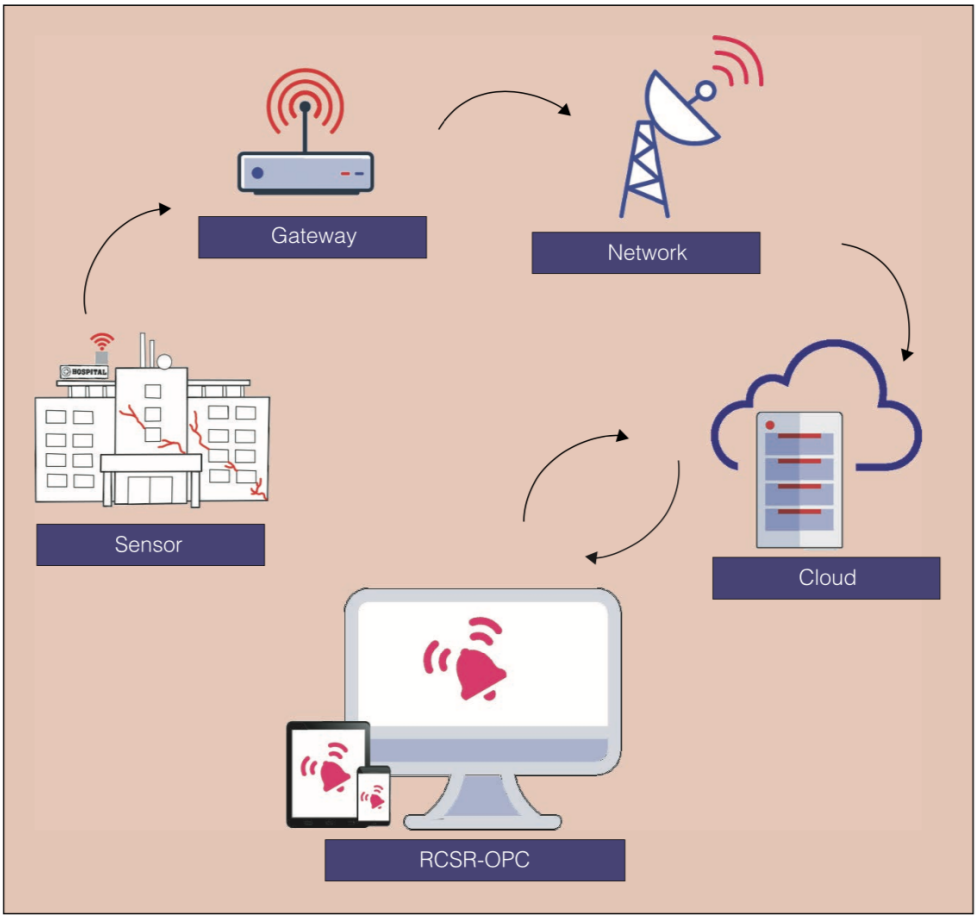

Figura 2.3: Schema di un sistema IoT di monitoraggio strutturale

Fonte: [9]

Il gateway è un nodo posto a lato della rete di monitoraggio che comunica sia con la rete di sensori che con il sottosistema tramite Internet. Esso è in grado di gestire ed ottimizzare le richieste di dati, le notifiche degli eventi, controllare la connettività dei nodi sensori e eseguire i test di integrità del sistema implementando i protocolli IoT. Il database locale incorporato consente al sistema di archiviare una quantità di dati utile a risolvere i problemi

di connessione remota ed evitare la perdita di dati.

Dal remote control è possibile conoscere lo stato di tutti i nodi di sensori connessi ed eseguire richieste specifiche a determinati sensori. Inoltre il RCSR è responsabile della memorizzazione dei dati prodotti da tali sensori. Il database utilizzato in genere al giorno d'oggi è un cloud database in quanto è in grado di memorizzare una grande mole di dati che saranno successivamente oggetto di analisi. L'RCSR in genere è connesso ad un OPC server utilizzato per interagire con i sistemi industriali standard [9].

Il monitoraggio strutturale con sistemi IoT è un approccio efficace alla manutenzione regolare di ponti, edifici, strade e altre strutture. Implementando tecnologie wireless che non richiedono il cablaggio della rete è possibile ridurre i tempi di sviluppo del sistema di monitoraggio; in questo modo è possibile integrare il sistema in strutture nuove ed esistenti, con un funzionamento a bassa invasività. Gli effetti positivi immediati di un IoT-SHM sono la drastica riduzione dei costi di monitoraggio e la crescente sicurezza a causa del monitoraggio continuo.
Inoltre l'integrazione di sistemi di monitoraggio IoT nelle smart city e nelle smart house consentirà l'ottimizzazione del monitoraggio su vaste aree e non solo su singole strutture.

2.2 Web of Things

Come descritto nella sezione 2.1 l'ecosistema IoT presenta alcune limitazioni, in particolare le difficoltà emergono nel momento in cui si ha la necessità di voler integrare una molteplicità di dispositivi e di aziende diverse all'interno di un unico sistema. Tale problema è strettamente correlato all'eterogeneità di soluzioni proposte dal mercato: il continuo avanzamento elettronico ha portato alla nascita di un numero sbalorditivo di dispositivi diversi e spesso incompatibili tra loro, inoltre sia i venditori che gli stessi organismi di standardizzazione hanno proposto diversi standard nel corso degli anni, ma nessuno di essi è stato così preponderante da guidare gli altri verso un'unica direzione per permettere lo sviluppo di una 'soluzione universale'. Ed è proprio per questo motivo che a volte l'utente finale si vede costretto ad affidarsi e ad usufruire delle soluzioni e dei prodotti offerti da un'unico produttore al fine di avere un'interoperabilità all'interno del proprio sistema.

Da ciò ne deriva che la scelta di adottare solamente le soluzioni proposte da un unico vendor può comportare delle limitazioni e delle mancanze alle funzionalità necessarie.

Tutto questo ha portato all'apertura verso il Web of Things (WoT), in cui il concetto di Thing è stato virtualizzato e trattato come proxie per realtà fisiche e astratte, e il concetto di Web si riferisce all'idea che queste Thing sono accessibili tramite tecnologie Web, quali appunto HTTP a livello di protocollo e API di scripting a livello applicativo [14].
Infatti il Web of Things si concentra esclusivamente sui protocolli e servizi del livello Application del modello OSI.

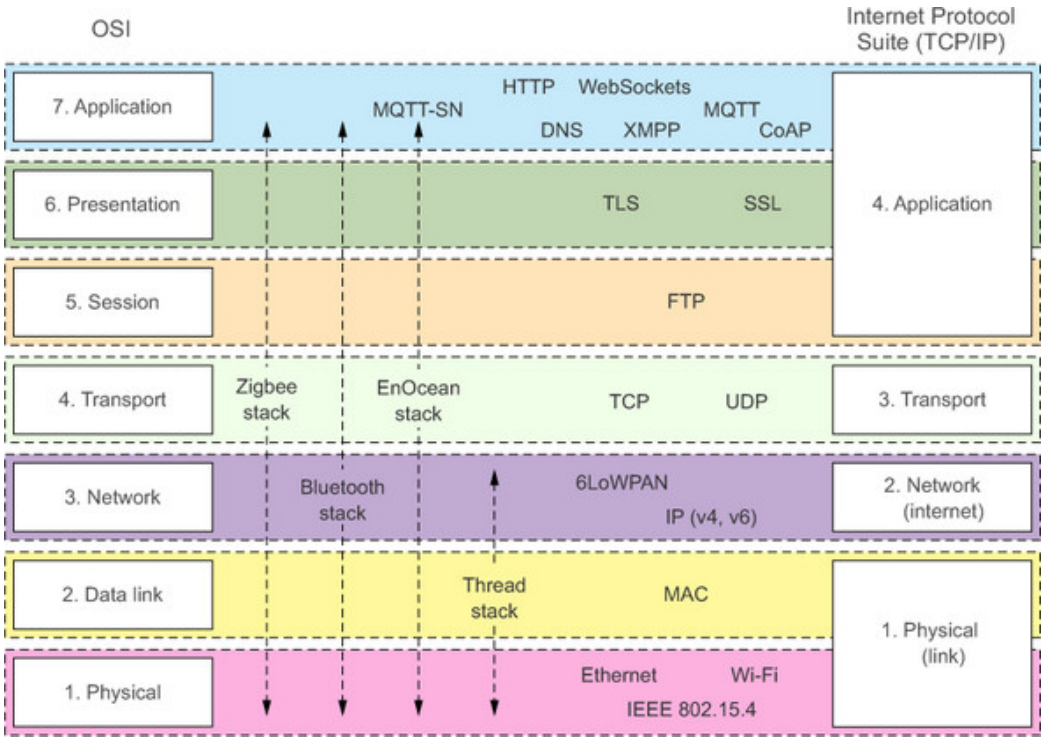

Figura 2.4: Confronto tra modello OSI e Internet Protocol Suite (TCP/IP)

Fonte:[15]

Il modello architetturale del WoT non è composto da rigidi strati come il modello OSI, visibile in figura 2.4, ma è diviso in livelli, i quali permettono di estendere le funzionalità (figura 2.5). I livelli sono delle astrazioni in cui ciascuno di essi si basa sul livello successivo; un livello serve il livello sopra ed è servito dal livello sottostante. Ciò significa che un protocollo in un determinato livello può solo fare ipotesi su ciò che offrirà il livello direttamente sotto. Di conseguenza, ogni livello si concentra su una particolare serie di problemi e risolve tale problema per i livelli sopra.

Il livello Applicazione è dove viene implementata l'architettura della Web of Things. Ovviamente, ciò significa che il WoT non può esistere senza i livelli di rete sotto il livello Applicazione. Per utilizzare i dati del mondo reale

dalle applicazioni su smartphone e sui browser, dobbiamo prima ottenere in qualche modo questi dati dai livelli inferiori del modello. Quindi vi è un'astrazione dei livelli sottostanti al livello Applicazione, perciò non importa quali protocolli o interfacce vengano utilizzati al di sotto.

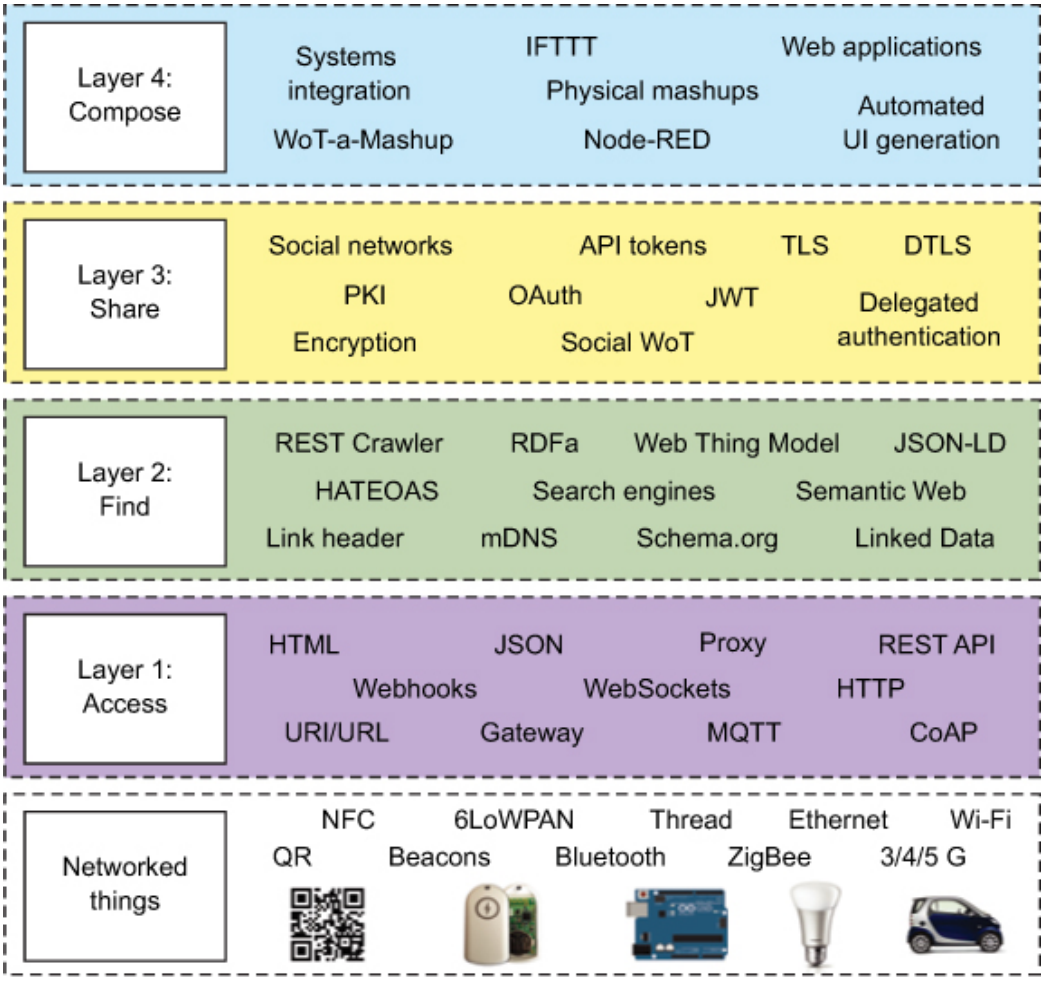

Figura 2.5: Stack architetturale del Web of Things

Fonte:[15]

Ora andremo ad analizzare i livelli che compongono lo stack architetturale del WoT:

- **Livello 1: Access**
  Il livello Access è il livello più importante perchè ha il compito di connettere la Thing ad Internet tramite un'API web. Questo livello è responsabile di trasformare una qualsiasi Thing in una Web Thing programmabile con cui altre applicazioni e dispositivi possono facilmente comunicare. Le Thing possono esporre i loro servizi tramite diverse modalità: tramite HTTP, mediante l'utilizzo di una RESTful API e JSON come formato di dati, tramite WebSocket o MQTT [1] nel caso si necessiti di una comunicazione real-time o even-drive.
  Però non tutte le Thing sono in grado di comunicare direttamente tramite protocolli web, ma ciò non esclude che possano far parte del WoT. Come vedremo nella sezione 3.1 esistono pattern specifici utili a gestire tali integrazioni.

- **Livello 2: Find**
  L'obiettivo del secondo livello è di fornire un modo per localizzare la Thing nel web. L'approccio è di riutilizzare gli standard del semantic web per descrivere le Thing e i loro servizi. Ciò consente la ricerca di Thing attraverso i motori di ricerca e altri indici web, nonché la generazione automatica di interfacce utente o tools per interagire con le Thing.

- **Livello 3: Share**
  Una delle idee su cui si basa il Web of Things è quella della Thing che "pubblica" dati sul web, sui quali possono essere applicate tecniche di big-data analysis. La responsabilità del livello Share è di specificare come i dati generati dalle Thing possano essere condivisi in modo efficiente e sicuro.

[1] `http://mqtt.org`

- **Livello 4: Compose**
  Una volta che le Thing sono sul web (livello 1) dove possono essere trovate da macchine ed umani (livello 2) e le loro risorse possono essere condivise in modo sicuro (livello 3), il livello Compose ha l'obiettivo di integrare i dati provenienti da Thing eterogenee all'interno di un ecosistema di applicazioni come software di analisi e applicazioni di monitoraggio. Gli strumenti del livello Compose possono spaziare dai web toolkit (Javascript SDK), a dashboard con widget programmabili, che verranno approfondite nel capitolo 4, fino ad intere piattaforme di mashup come Node-RED. Questi strumenti possono consentire alle persone di creare applicazioni utilizzando i servizi WoT senza richiedere competenze di programmazione [15].

2.3 W3C WoT

Il World Wide Web Consortium (W3C) è una community internazionale in cui i membri del gruppo, uno staff a tempo pieno e il pubblico, lavorano insieme per sviluppare gli standard Web [1].

In particolare all'interno del W3C è stato istituito un working group specifico per lavorare alla standardizzazione del Web of Things. L'obiettivo del gruppo è quello di contrastare la frammentazione dell'IoT fornendo vari componenti standard complementari (es Metadata e API) che consentono una facile integrazione tra piattaforme IoT e domini applicativi [16]. Il lavoro eseguito dal gruppo ha portato alla nascita di diversi documenti: 2 documenti normativi che devono essere seguiti fedelmente al fine di soddisfare i requisiti dello standard, e 3 documenti informativi utile all'utente alla comprensione dei concetti presenti nei documenti normativi.

I 2 documenti normativi sono:

- **WoT Architecture**
  Questa specifica definisce l'architettura di alto livello per i singoli blocchi di costruzione WoT e le configurazioni della piattaforma necessarie. Inoltre, identifica gli scenari di distribuzione considerati dal Web of Things.

- **WoT Thing Description**
  Questo documento descrive un modello formale e una rappresentazione comune per una descrizione semantica delle Thing del Web of Things (WoT). Il Thing Description costituisce la parte più importante, in quanto fornisce un insieme di interazioni basate su un piccolo vocabolario che permette di integrare dispositivi diversi consentendo ad applicazioni diverse di interagire. Inoltre descrive i metadati rispetto alla sicurezza e alla comunicazione.

I 3 documenti informativi sono i seguenti:

- **WoT Scripting API**
  Una suite di API per gli sviluppatori di applicazioni per il Web of Things. Rappresenta l'interfaccia WoT per gestire l'interazione tra Thing nonchè il loro lifecycle.

- **WoT Binding Templates**
  Fornisce una serie di pattern ed estensioni del vocabolario che consentono di adattare una Thing Description ad un protocollo specifico, o all'utilizzo di strutture di payload di dati attraverso i diversi standard.

- **WoT Security and Privacy Guidelines**
  Specifica i requisiti di sicurezza generali per un sistema Web of Things (WoT) utilizzando un "threat model". Il WoT threat model definisce le principali parti interessate alla sicurezza, le risorse rilevanti, i possibili attaccanti, i punti di attacco e infine le minacce per un sistema WoT.

Questi documenti sono stati pubblicati per la prima volta nel settembre 2017, e diventeranno della W3C Recommandation entro Marzo 2020[16].

Al giorno d'oggi all'interno del working group sono presenti oltre 100 partecipanti, infatti oltre ai membri W3C collaborano numerose aziende, quali Huawei, Alibaba Group, Fujitsu Ltd., Hitachi Ltd., Intel Corporation, Mitsubishi Eletric Corporation, Oracle Corporation, Panasonic e altre [17].

Capitolo 3

W3C WoT Architecture

L'architettura del WoT proposta dal W3C ha lo scopo di consentire l'interoperabilità tra piattaforme IoT e domini applicativi. L'obiettivo del WoT è di preservare ed integrare gli standard e le soluzioni IoT esistenti, infatti in generale l'architettura W3C WoT è progettata per descrivere ciò che già esiste piuttosto che per prescrivere cosa implementare.
L'architettura astratta WoT definisce un quadro concettuale di base che può essere mappato su diversi scenari di deployment concreti, identificando una serie di componenti che possono essere integrati e correlati tra loro a seconda dei requisiti del caso d'uso in questione.
Va sottolineato che i documenti presi in considerazione sono in continua evoluzione, pertanto i contenuti descritti fanno riferimento all'ultima versione del documento WoT Architecture [18] rilasciato il 31 Gennaio 2020 dal W3C.

3.1 Casi d'uso

L'architettura Web of Things non pone alcuna limitazione ai casi d'uso e ai domini delle applicazioni. I casi d'uso presi in oggetto per raccogliere modelli comuni utili a costruire l'architettura astratta sono i seguenti:

- Consumer
- Industriale: Smart factories

  - Monitoraggio avanzato in tempo reale delle apparecchiature di produzione
  - Prevenzione di problemi di produttività
  - Controllo di emissioni
  - Ottimizzazione della catena di approvvigionamento
  - Lettura automatica di contatori residenziali, industriali e commerciali

- Trasporto e logistica
  - Monitoraggio dei veicoli
  - Tracciamento delle spedizioni
  - Controllo qualità delle merci e scorte di magazzino

- Agricoltura
  - Monitoraggio delle condizioni del suolo per la creazione di piani ottimali per l'irrigazione e la concimazione.
  - Monitoraggio delle condizioni del prodotto per ottimizzare la qualità e la produzione di prodotti agricoli.

- Sanità
  - Raccolta ed analisi di dati degli studi clinici come base di approfondimento per nuove aree di studio.
  - Monitoraggio remoto di pazienti per ridurre il rischio di situazioni critiche non rilevate per i pazienti post ricovero ospedaliero.

- Monitoraggio ambientale
  - Controllo dell'inquinamento atmosferico, idrico, e fattori di rischio ambientale quali radioattività, polveri sottili, ecc.

- Smart cities

  - Monitoraggio di ponti, dighe, deterioramento materiali
  - Smart parking per l' ottimizzazione dei parcheggi
  - Controllo smart dell'illuminazione

- Smart Buildings
  - Monitoraggio del consumo energetico nell'edificio per la riduzione degli sprechi

- Automobili
  - Monitoraggio dello stato operativo per prevenire la manutenzione e ridurre i costi
  - Migliorare la sicurezza dell'utente attraverso sistemi di notifica su condizioni stradali critiche e fattori di rischio.

Inoltre vengono definiti anche pattern di casi d'uso comune che illustrano come dispositivi/thing interagiscono con controller, altri dispositivi, agenti e servers. Tra questi pattern, partendo dai più semplici, abbiamo i device controllers e i thing-to-thing, fino ad arrivare a pattern più complessi come Digital twin e cross-domain collaboration. Un digital twin è una rappresentazione virtuale, ovvero un modello di uno o più dispositivi che risiede in un server cloud o in un edge device. Invece il cross-domain collaboration fornisce un esempio di come sistemi di domini differenti (come smart cities e smart factories) possano essere uniti all'interno di un unico ecosistema.

3.2 Architettura

3.2.1 Panoramica generale

I concetti di W3C WoT sono applicabili a tutti i livelli rilevanti per le applicazioni IoT:

- livello dispositivo
- livello edge
- livello cloud

La figura 3.1 fornisce una panoramica di come tali concetti possano essere applicati e combinati nei vari casi d'uso.

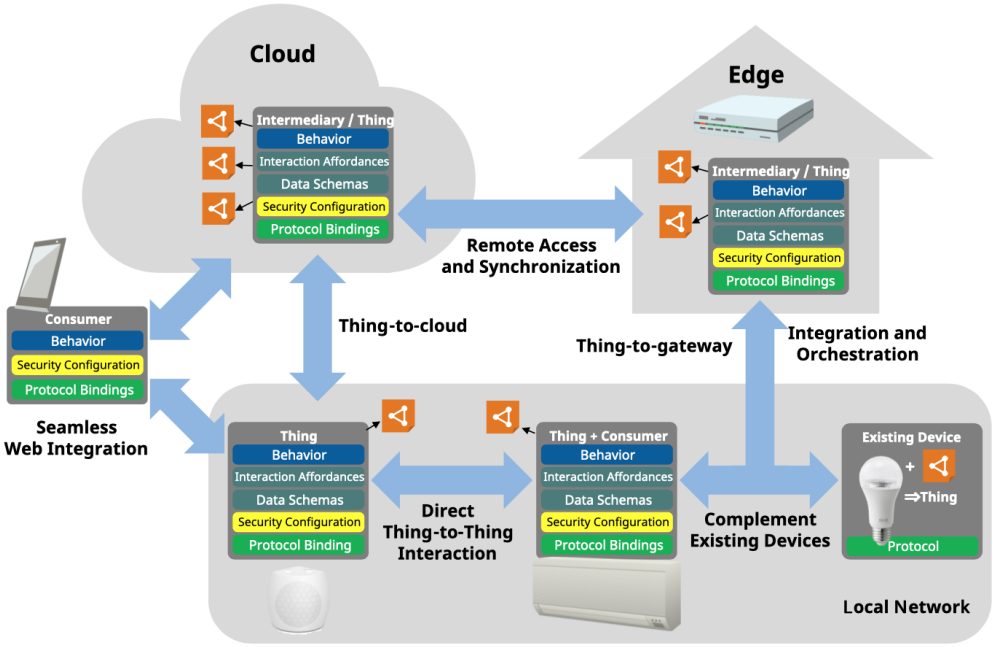

Figura 3.1: Architettura astratta del W3C WoT

Fonte:[18]

3.2.2 Thing

Una Thing è un'astrazione di un'entità fisica o virtuale (es. un dispositivo o una stanza) la quale è descritta attraverso metadati standardizzati. Nel W3C WoT tali metadati rappresentano la WoT Thing Description (TD). Il Consumer deve essere in grado di parsare e processare il formato di rappresentazione del TD, basato su JSON[2]. Una TD è un'istanza specifica (ovvero descrive una singola Thing, non tipi di Thing) che descrive attraverso una rappresentazione testuale (Web) una Thing, specificandone l'identificativo, le funzioni, gli attributi e i protocolli di comunicazione.

Per essere una Thing deve essere disponibile almeno una Thing Description. Attraverso la TD, essendo un formato standardizzato, i consumer sono in grado di scoprire e interpretare le capacità di una Thing (attraverso le annotazioni semantiche) e di adattarsi a diverse implementazioni quando interagiscono con essa, consentendo così un'interoperabilità tra diverse piattaforme IoT, ovvero standard ed ecosistemi. (Figura 3.2)

Figura 3.2: Interazione Consumer-Thing

Fonte:[18]

Una Thing può essere anche l'astrazione di un'entità virtuale. Un'entità virtuale è una composizione di più Thing (es. una stanza composta da più sensori). Un' opzione per tale scenario è quella di fornire un'unica Thing Description TD che funge da collegamento alle Thing secondarie gerarchiche

[2] https://www.json.org/

all'interno della composizione: in questo modo la TD agisce come punto d'ingresso e contiene solo i metadati generali e le proprietà globali. Tale collegamento non si applica solo a Thing gerarchiche, ma anche alle relazioni tra Thing e ad altre risorse generali del Web (Figura 3.3).

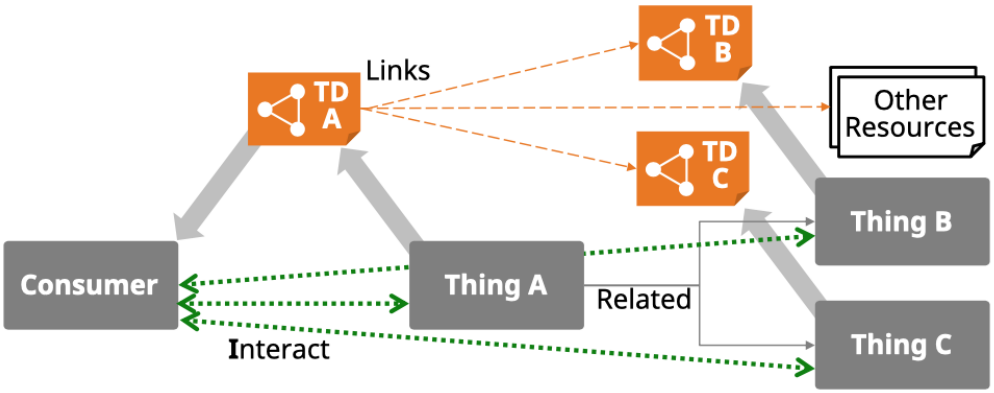

Figura 3.3: Collegamento tra Thing

Fonte:[18]

Le Thing devono essere ospitate su componenti di sistema in rete con un software stack che permetta l'interazione attraverso l'interfaccia di rete, ovvero la WoT Interface di una Thing. Un tipico esempio è un server HTTP in esecuzione in un dispositivo embedded con sensori ed attuatori, che si interfaccia con l'entità fisica attraverso l'astrazione della Thing. In uno scenario in cui le reti locali non sono raggiungibili tramite Internet, per ovviare a questa sistuazione il W3C WoT mette a disposizione degli intermediari tra il Consumer e la Thing (Figura 3.4).

Gli intermediari agiscono come proxies per le Thing, e dispongono di una TD simile alla Thing originale. Essi possono anche aumentare il numero di Thing esistenti con proprietà aggiuntive oppure comporre una nuova Thing tra quelle disponibili, formando così una nuova entità virtuale. Per i Consu-

mers gli intermediari vengono visti come una vera e propria Thing, in quanto dispone sia di un TD sia di una WoT Interface, perciò sono indistinguibili dalle Thing all'interno di un'architettura a strati come il Web.

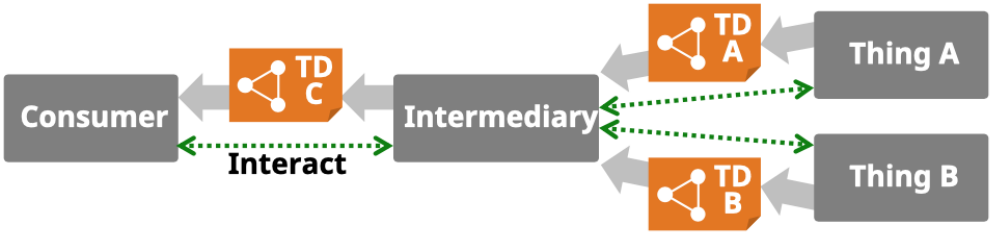

Figura 3.4: Intermediari tra Consumer e Thing

Fonte:[18]

Analizzando l'architettura interna di una Thing, visibile in figura 3.5, possiamo affermare che essa ha 4 aspetti architetturali d'interesse:

- **Comportamento**
  Il comportamento di una Thing include sia il comportamento autonomo che il gestore delle affordance d'interazione.

- **Affordance d'interazione**
  Le affordance forniscono un modello per far si che il Consumer possa comunicare con la Thing attraverso operazioni astratte.

- **Protocol bindings**
  Il protocol bindings aggiunge dettagli necessari a configurare ciascuna affordance d'interazione ad un messaggio concreto di un determinato protocollo.

- **Configurazione di sicurezza**
  La configurazione di sicurezza rappresenta il meccanismo utilizzato per controllare l'accesso alle affordance di interazione e la gestione della sicurezza dei metadati pubblici e dei dati privati.

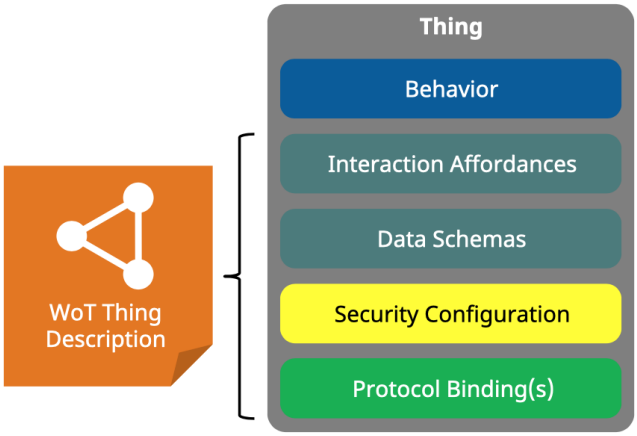

Figura 3.5: Aspetti architetturali di una Thing

Fonte:[18]

3.2.3 Modello di interazione

Il modello di interazione del W3C WoT introduce un'astrazione intermedia per formalizzare la mappatura dall'intento di applicazione a operazioni di protocollo concrete, inoltre impone delle limitazioni su come le affordance d'interazioni possano essere modellate.
Le Thing offrono 3 tipi di affordance d'interazione:

- **Proprietà**
  Espone uno stato della Thing, il quale deve essere accessibile dal consumer e opzionalmente può essere anche sovrascrivibile. Esempi di proprietà sono i valori dei sensori (solo lettura), i parametri di configurazione (lettura-scrittura) o lo stato di una Thing (lettura-scrittura).

- **Azioni**
  Permettono di invocare funzioni all'interno della Thing. Tali azioni possono manipolare una o più proprietà descritte sopra. Esempi di azioni sono la modifica di più proprietà contemporaneamente, come l'intensità della luminosità di una luce.

- **Eventi**
  Permettono di trasferire dati in modo asincrono tra il Consumer e la Thing. Gli eventi possono essere attivati attraverso delle condizioni che non sono esposte nelle proprietà. Alcuni esempi sono eventi come un allarme o elementi di una serie temporale che vengono inviati regolarmente.

3.2.4 Servient e interconnettività

Quando i componenti astratti dell'architettura WoT vengono implementati all'interno di un software stack, quest'ultimo viene chiamata **Servient**. Una servient può implementare una Thing, e ne contiene la sua rappresentazione, chiamata *Exposed Thing*, esponendo quindi l'interfaccia della Thing al consumer. Il consumer, affinchè possa consumare la TD della Thing, deve essere implementato da una Servient.

Nel caso del consumer, una servient fornisce la rappresentazione di una Thing chiamata *Consumed Thing* e la rende disponibile alle applicazioni che sono in esecuzione sul software stack della Servient che necessitano di elaborare la TD per interagire con le Thing. Anche l'intermediario, uno dei componenti dell'architettura WoT, può essere implementato mediante una Servient. L'intermediario si posiziona tra il Consumer e la Thing ed assume il compito sia di consumare la Thing che allo stesso tempo di esporre la Thing per il Consumer (Figura 3.6).

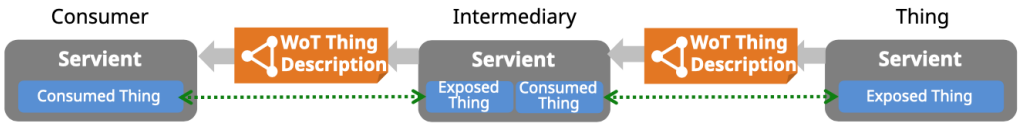

Figura 3.6: Esempio di Servient come intermediario

Fonte:[18]

Quindi la comunicazione tra Consumer e Thing può avvenire in maniera diretta, quando entrambi i Servient utilizzano lo stesso protocollo di rete, oppure tramite un intermediario nel caso di protocolli di rete differenti o se i Servient si trovano su reti differenti. Ad esempio, l'intermediario potrebbe utilizzare il protocollo COAP per comunicare con la Thing ed affidarsi ad HTTP per comunicare con il consumer.

3.3 Elementi principali

Gli elementi principali (Building blocks) del Web of Things consentono di passare dall'architettura astratta all'implementazione. Di seguito andremo ad analizzarli nel dettaglio.

3.3.1 WoT Thing Description

Il WoT Thing Description [20] è un documento che definisce un modello di informazione basato su un vocabolario semantico, rappresentato in un formato basato su JSON. Sia il modello che la rappresentazione del TD sono allineati con i Linked Data [2], in questo modo le varie implementazioni possono scegliere di usare JSON-LD [11] e i database a grafo per consentire una potente elaborazione semantica dei metadati.

Una Thing Description (TD) descrive l'istanza di una Thing, ed ha quattro componenti principali: metadati testuali sulla Thing stessa (come nome, ID e descrizione), un set di Affordance di interazione che indicano in che modo la Thing può essere utilizzata, metadati relativi allo schema dati, alla comunicazione, link e ai meccanisimi di sicurezza. La TD è costruita attorno al modello di interazione descritto nel paragrafo 3.2.3 ed è in grado di supportare più paradigmi di comunicazione (request-response, publish-subscribe, ecc.). La Thing Description (TD) viene considerata il punto di accesso ad una Thing per conoscere i servizi a disposizione e le risorse correlate. Può essere vista come l' *index.html* in un sito Web.

```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "id": "urn:dev:ops:32473-WoTLamp-1234",
    "title": "MyLampThing",
    "securityDefinitions": {
        "basic_sc": {"scheme": "basic", "in":"header"}
    },
    "security": ["basic_sc"],
    "properties": {
        "status" : {
            "type": "string",
            "forms": [{"href": "https://mylamp.example.com/
    status"}]
        }
    },
    "actions": {
        "toggle" : {
            "forms": [{"href": "https://mylamp.example.com/
    toggle"}]
        }
    },
    "events":{
        "overheating":{
            "data": {"type": "string"},
            "forms": [{
                "href": "https://mylamp.example.com/oh",
                "subprotocol": "longpoll"
            }]
        }
    }
}
```

Listing 3.1: Esempio di Thing Description in JSON-LD 1.1.

Una TD, idealmente, dovrebbe essere creata e ospitata dalla Thing stessa e recuperata tramite ricerca. Tuttavia può essere anche ospitata esternamente nel momento in cui una Thing presenta delle restrizioni (e.g. capacità di

memoria limitata, potenza limitata). Un pattern comune per migliorare la ricerca e facilitare la gestione dei dispositivi consiste nel salvare la TD all'interno di una directory che agisca da cache. In questo modo i consumers dovrebbero salvare la TD nella cache la prima volta che si connettono alla Thing, modificandola solamente nel caso di un aggiornamento.
Per avere la massima interoperabilità semantica, i TD dovrebbero fare uso del vocabolario specifico dello standard. Tuttavia lo sviluppo di un vocabolario specifico per un determinato dominio non rientra attualmente nell'ambito dell'attività di standardizzazione WoT del W3C.

Nel complesso, la WoT Thing Description promuove l'interoperabilità in due modi:

- Consente la comunicazione machine-to-machine nel WoT.
- Può 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 propri dati.

3.3.2 WoT Binding Templates

L'IoT utilizza una varietà di protocolli per l'accesso ai dispositivi, in quanto non esiste un unico protocollo adatto a tutti i contesti. Per questo, una delle sfide principali del WoT è quella di riuscire ad integrare le varie piattaforme IoT (e.g. OCF [3], oneM2M [4], Mozilla IoT[5] , ecc.) e i dispositivi RESTful che non seguono un particolare standard ma dispongono di un'interfaccia d'accesso HTTP o COAP. WoT sta affrontando questa varietà attraverso i Protocol Bindings, che devono soddisfare una serie di vincoli.

La specifica del WoT Binding Templates [8] fornisce una collezione di "blueprint" di comunicazione che contengono indicazioni su come interagire

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

con diverse piattaforme IoT. Quando si descrive un particolare dispositivo o servizio IoT, è possibile utilizzare il Binding Templates per la piattaforma IoT corrispondente per cercare i metadati di comunicazione che devono essere aggiunti nella Thing Description per supportare quella specifica piattaforma. La figura 3.7 mostra come vengono applicati i Binding Templates.

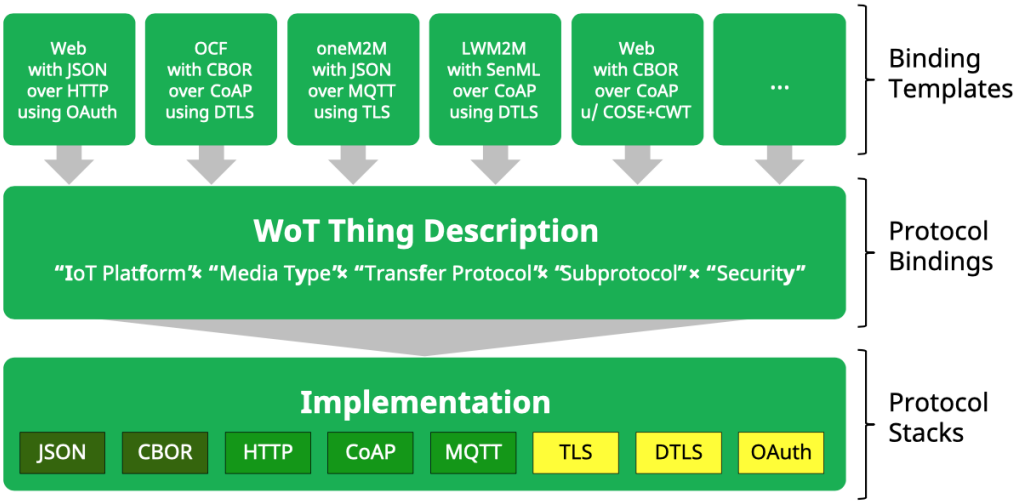

Figura 3.7: Dai Binding Templates ai Protocol Binding

Fonte:[18]

Un WoT Binding Templates viene creato una sola volta per ciascuna piattaforma IoT e può quindi essere riutilizzato in tutte le TD per i dispositivi di quella piattaforma. I Consumers che consumano una TD devono implementare il corrispondente Binding Template includendo lo stack di protocollo configurato in base alle informazioni fornite nel Thing Description.

I metadati del Protocol Bindings comprendono cinque dimensioni:

- **IoT Platform**
  Le piattaforme IoT introducono spesso modifiche proprietarie a livello di applicazione come header HTTP specifici della piattaforma o opzioni CoAP.

- **Media Type**
  Le piattaforme IoT spesso differiscono nei formati di rappresentazione (o nelle serializzazioni) utilizzati per lo scambio di dati. I Media Type permettono di identificare questi formati.

- **Transfer Protocol**
  Comprendono i protocolli standard dello strato Applicazione (HTTP, MQTT, CoAP, WebSocket, ecc.). Lo schema URI del form di invio contiene le informazioni richieste per identificare il protocollo di trasferimento

- **Subprotocol**
  I Transfer Protocol possono avere meccanismi di estensione che devono essere noti per interagire correttamente. Tali sotto-protocolli non possono essere identificati dal solo schema URI ma devono essere dichiarati esplicitamente. Un esempio di sotto-protocollo è il "long polling" per HTTP.

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

3.3.3 WoT Scripting API

La specifica della WoT [12] Scripting API definisce la struttura e gli algoritmi dell'interfaccia di programmazione che consente agli script di scoprire, recuperare, consumare, produrre ed esporre le WoT Thing Description.
Il sistema di runtime del WoT Scripting API crea un'istanza di oggetti locali che fungono da interfaccia per altre Thing e le loro convenzioni di interazione (proprietà, azioni ed eventi). Consente inoltre agli script di esporre Thing, ovvero definire e implementare le interazioni e pubblicare una Thing Description. Lo scripting è un componente facoltativo di "convenienza" in WoT ed

è generalmente utilizzato in gateway più potenti in grado di eseguire un runtime WoT e la gestione degli script. Queste interfacce descrivono un'API ECMAScript di basso livello che segue le specifiche della TD con precisione. Possono essere utilizzati anche altri linguaggi e runtime, l'importante è che rispettino i vincoli.
Nel WoT Scripting API vi sono 3 componenti fondamentali:

- **L'oggetto WoT**
  Definisce il punto di ingresso API esposto come singleton e contiene i metodi API. Esso non espone proprietà, ma solo metodi per consumare, scoprire ed esporre Thing.

- **Consumed Thing Interface**
  Rappresenta l'API Client per consumare le Thing.

- **Exposed Thing Interface**
  L'interfaccia ExposedThing è l'API del server per gestire la Thing che consente di definire gli handler per le richieste, le Proprietà, le Azioni e gli Eventi.

3.4 Implementazione della Servient

Come definito nel paragrafo 3.2.4 una Servient è uno stack software che implementa i componenti WoT descritti nella sezione 3.3. Una Servient può eseguire, esporre o consumare più Thing contemporaneamente, infatti, a seconda del protocol binding, possono agire sia da client sia da server.

In figura 3.8 viene rappresentata nel dettaglio la composizione di una Servient.

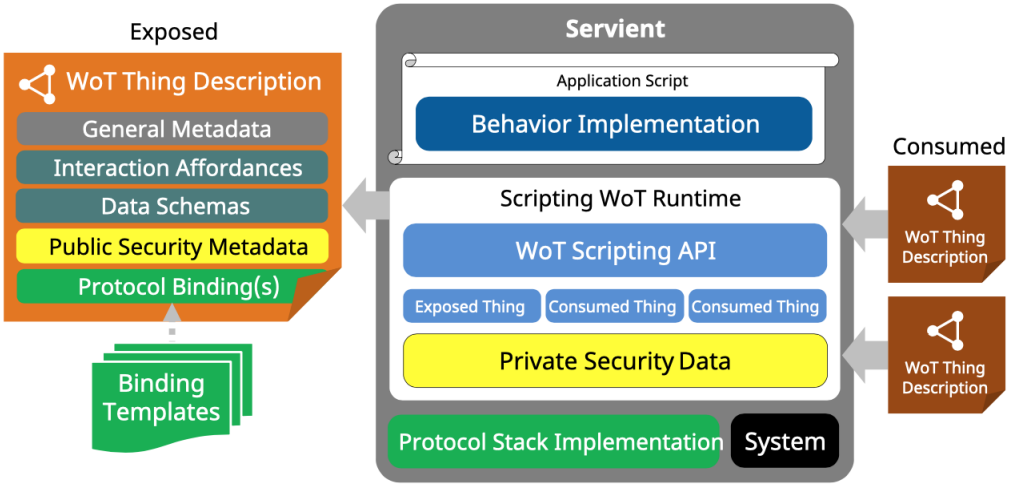

Figura 3.8: Implementazione di una Servient

Fonte:[18]

Alcuni dei moduli presenti nella servient sono già stati analizzati nella sezione precedente, ma mancano ancora alcuni moduli utili ad avere una panoramica completa dell'implementazione di una Servient [19].

- **Behavior Implementation**
  Il comportamento (Behavior) definisce la logica applicativa di una Thing. L'implementazione di esso all'interno di una Servient definisce quali Thing, Consumer e Intermediari sono ospitati su questo componente. In linea di principio, è possibile utilizzare qualsiasi linguaggio di programmazione e API per definire il comportamento di una Thing, purchè i modelli di interazione siano esposti esternamente tramite un'interfaccia WoT. L'adattamento tra API e Protocol stack è gestito dal WoT Runtime.

- **WoT Runtime**
  Tecnicamente, l'astrazione della Thing e il suo modello di interazione vengono implementati in un sistema runtime. Il WoT Runtime fornisce l'ambiente di esecuzione per l'implementazione del behavior e permet-

te di esporre o consumare una Thing, perciò deve essere in grado di ricercare e processare la WoT Thing Description. Le implementazioni delle applicazioni avvengono solitamente tramite script in Javascript. Inoltre sempre il WoT Runtime si interfaccia sia con l'implementazione del protocol stack della Servient, separando i dettagli dei Protocol Bindings dall'implementazione del behavior e sia con la System API per la gestione dell'hardware locale.

- **Protocol Stack Implementation**
  Il protocol stack di una Servient implementa l'interfaccia WoT delle Thing esposte e viene utilizzato dai Consumers per accedere alle interfacce remote delle Thing. Genera i messaggi di protocollo concreti per interagire con le Thing attraverso la rete, inoltre supporta più Protocol Bindings per garantire l'interazione con una molteplicità di piattaforme IoT. In molti casi, quando vengono utilizzati dei protocolli standard, è possibile utilizzare degli stack di protocolli generici. In questo caso i metadati della TD vengono utilizzati per selezionare e configurare quello corretto. Inoltre i parser e i serializzatori per i vari formati (identificati tramite i Media Type) possono essere condivisi tra i vari stack.

- **System API**
  Un'implementazione del WoT Runtime permette di accedere all'hardware locale o ai servizi di sistema attraverso API proprietarie. Un dispositivo può trovarsi fisicamente esterno alla Servient, ma connesso attraverso protocolli proprietari. In questo caso il WoT Runtime può avere accesso al dispositivo tramite le API del protocollo proprietario, per poi esporlo come Thing attraverso le Scripting API. Una Servient può agire come gateway per il dispositivo, ma questa opzione va considerata solo nel momento in cui non è possibile descrivere il dispositivo attraverso una TD.

Capitolo 4

Tool per la gestione e visualizzazione dati IoT

Al giorno d'oggi, come già accennato nel capitolo 1, l'IoT dispone di più di 30 miliardi dispositivi connessi. Ciò che crea valore in questi numeri ed ha un impatto diretto sul mercato economico è l'acquisizione continua dei dati prodotti da tali dispositivi. Infatti analizzando questi dati le aziende possono identificare colli di bottiglia nell'approvvigionamento, prevedere guasti alle apparecchiature e ottimizzare i costi operativi.

Tuttavia si rileva la presenza del problema seguente: i dati generati dai dispositivi nel mondo sono ampiamente sottoutilizzati. Ciò è dovuto al fatto che non solo l'acquisizione, l'archiviazione e l'analisi di dati IoT sono piuttosto costose, ma le aziende spesso hanno una mancanza di strumenti ed esperti che riescono ad interpretare i dati in modo efficiente.

È per questo che le tecnologie di gestione e visualizzazione dati assumono un ruolo importante nell'ecosistema IoT, in quanto senza questi tool sarebbe praticamente impossibile interpretare i valori provenienti da più sensori distribuiti in una rete, filtrare le entità non significative e identificare real-time i cambiamenti.

Però va sottolineato come il concetto di gestione e visualizzazione dei dati vada oltre la semplice visualizzazione di informazioni in grafici e diagrammi leggibili agli utenti umani, in quanto gli input di dati generati dai sensori possono essere usati dall'applicazione come trigger per l'esecuzione automatica di azioni.
In questo modo s'incrementa l'automazione, che costituisce l'obiettivo primario dell'Internet of Things.

Di seguito vengono analizzate e successivamente confrontate le caratteristiche e proprietà delle principali piattaforme di visualizzazioni dati Iot open source presenti sul mercato.

4.1 Grafana

Grafana [4] è uno tool open source di visualizzazione e analisi dei dati specializzato nella visualizzazione di dati in time-series. Possiede un vasto range di tecniche di visualizzazione, permette di creare e gestire più dashboard contemporaneamente che supportano una varietà di widgets.
Originariamente Grafana è stato progettato per il monitoraggio dello stato della CPU e del sistema, ma come accennato in precedenza, lo strumento è meglio noto per essere uno dei migliori tool nel mercato per la visualizzazione dei dati in time-series, utile al monitoraggio di sensori in real-time provenienti dal mondo IoT.
Grafana permette una molteplicità di funzioni, come notifiche, alerts, filtri personalizzati ed annotazioni per lo streaming di dati. Supporta l'integrazione di più database, ed è possibile inoltre mixare dati provenienti da risorse differenti all'interno di un unico grafico, nonchè eseguire queries su tali dati e filtrare i risultati.

Figura 4.1: Dashboard di Grafana

Fonte:[4]

4.2 Kibana

Kibana [7] è un tool di visualizzazione che fa parte dello stack ELK[6]. ELK è un toolkit composto da 3 strumenti open source: Elasticsearch, Logstash e Kibana. Kibana, che costituisce la 'K' dello stack ELK, fornisce agli utenti uno strumento per esplorare, visualizzare e costruire dashboard sulla base dei dati di log memorizzati nei cluster Elasticsearch. Le caratteristiche principali di Kibana sono le queries su dati e l'analisi dei log. Gli utenti attraverso determinate funzioni possono cercare i dati indicizzati in Elasticsearch per eventi specifici o stringhe all'interno dei propri dati al fine di eseguire analisi e diagnostica. Sulla base di queste query, gli utenti possono utilizzare le funzionalità di visualizzazione di Kibana che consentono di visualizzare i dati in vari modi, utilizzando grafici, tabelle, mappe geografiche e altri tipi di visualizzazioni.

[6] `https://www.elastic.co`

Kibana non supporta l'integrazione con altre fonti di dati e funziona solo con Elasticsearch.

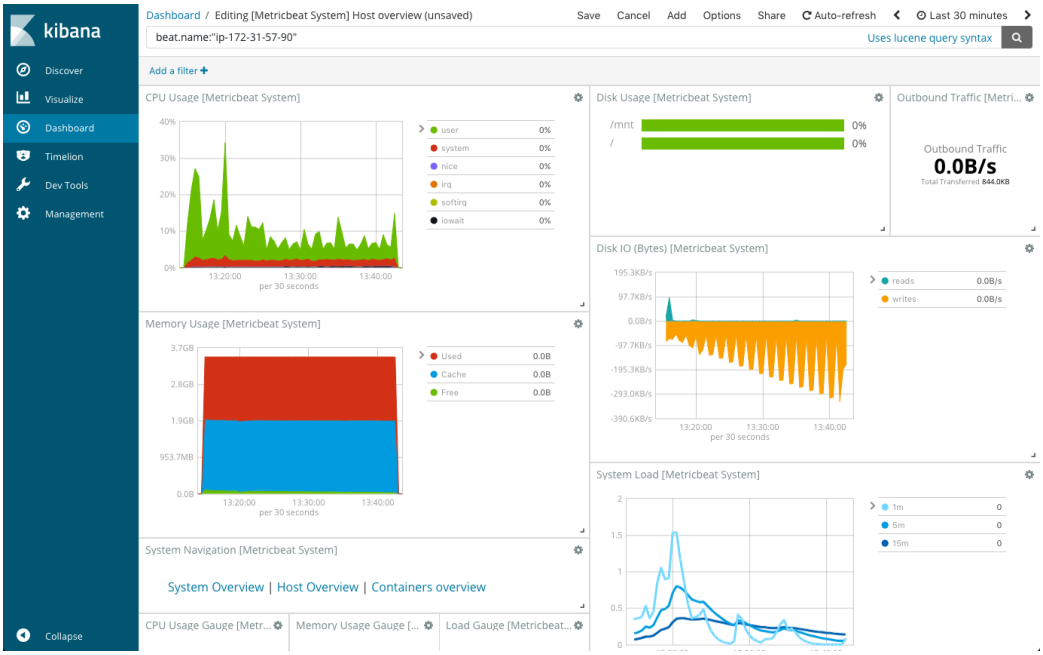

Figura 4.2: Dashboard di Kibana

Fonte:[7]

4.3 Proprietà a confronto

Prendendo in considerazione i due tool descritti sopra, Grafana e Kibana, ora verranno confrontate le proprietà, i punti di forza e debolezza che caratterizzano le piattaforme IoT di visualizzazione:

- **Visualizzazione**
  Sia Grafana che Kibana sono essenzialmente tool di visualizzazione quindi entrambi offrono una vasta gamma di funzionalità per creare grafici e dashboard. Però tra i due Grafana è più popolare per la produzione di grafici e dashboard visivamente accattivanti, inoltre Grafana

è nota per essere più personalizzabile e flessibile rispetto a Kibana. Ma allo stesso tempo, Kibana è più facile da configurare. L'unica differenza sostanziale è che Kibana permette di visualizzare testo sotto forma di grafici, mentre ciò non è permesso in Grafana.

- **Logging vs Monitoring**
  La differenza chiave tra i due strumenti di visualizzazione deriva dal loro scopo. Grafana è progettato sia per l'analisi e la visualizzazione di metriche quali CPU di sistema, memoria, utilizzo del disco ecc. e sia per la visualizzazione di dati time-series in real time. Però Grafana non consente di eseguire queries su dati full-text. Kibana, d'altra parte, viene eseguito su Elasticsearch e viene utilizzato principalmente per l'analisi dei messaggi di log.

- **Installazione e setup**
  Sia Grafana che Kibana sono facilmente installabili e configurabili. Entrambi i tools supportano Linux, MacOS, Windows e anche Docker.

- **Data source e integrations**
  Grafana, come accennato, è progettato per lavorare con una molteplicità di database, quali ad esempio Elasticsearch, InfluxDB, Graphite, MySQL, MongoDB, PostgreSQL e altri. Per ogni database, Grafana dispone di un editor di query specifico personalizzato per le caratteristiche e le capacità incluse in tali DB. Kibana, d'altra parte, è progettato per funzionare solo con Elasticsearch e quindi non supporta nessun altro tipo di database.

- **Authentication**
  Grafana fornisce dei meccanismi di autenticazione e controllo di accesso integrati che consentono di limitare e controllare l'accesso degli utenti alla dashboard. Inoltre, l'API di Grafana può essere utilizzata per attività come il salvataggio di una dashboard specifica, la creazione di utenti e l'aggiornamento di risorse dati.

Kibana non offre alcuna funzionalità di gestione degli utenti e di autenticazione. Pertanto chiunque abbia il link alla dashboard di Kibana può vedere i dati. Però in Kibana è possibile implementare le restrizioni di utenti attraverso applicazioni di terze parti.

- **Querying**
  Le queries e la ricerca dei logs sono il punto di forza di Kibana. Utilizzando la sintassi Lucene[7] e la query DSL Elasticsearch, è possibile cercare i dati memorizzati in Elasticsearch visualizzando i dati in ordine cronologico. Con Grafana, gli utenti utilizzano un Query Editor. Ogni risorsa proveniente da un DB differente ha un diverso query editor su misura, il che significa che la sintassi utilizzata varia in base all'origine dati. Quindi ad esempio le queries di Graphite saranno diverse dalle query di MySQL.

- **Alerts**
  Una differenza fondamentale tra Grafana e Kibana sono gli avvisi. Dalla versione 4.x in Grafana è presente un motore di avvisi incorporato che consente agli utenti di impostare regole nelle proprie dashboard, il cui risultato proveniente dall'avviso può essere triggerato ad un endpoint a scelta (e.g. generazione automatica di una mail). Kibana, al contrario, non dispone di alcuna funzionalità di avviso.

- **Community**
  Entrambi gli strumenti open source hanno una potente comunità di utenti e collaboratori attivi. Guardando i due progetti su GitHub, Kibana arriva a circa 31000 commits mentre Grafana "solo" 24000. Questo è un segno di come i progetti siano potenti ed in continuo aggiornamento.

[7] `https://lucene.apache.org`

Parte II

Mac4Pro: Gestione e visualizzazione dati IoT

Capitolo 5

Progettazione

In questo capitolo viene fatta una panoramica della fase di progettazione del sistema di gestione dati IoT realizzato. Inizialmente viene descritto il contesto all'interno del quale si posiziona il progetto. In seguito, dopo aver definito gli obiettivi nonchè i requisiti necessari, viene descritta la struttura del sistema. Vengono analizzati nel dettaglio ciascun componente del sistema, attraverso una descrizione di caratteristiche, funzionalità e ruoli all'interno dell'architettura.

5.1 Mac4Pro

Mac4Pro [13] è un abbreviazione del progetto "Manutenzione Intelligente di impianti industriali e opere civili mediante tecnologie di monitoraggio 4.0 e approcci prognostici", il cui ente finanziatore è l'INAIL.

Il tema centrale è la **prognostica**: l' obiettivo generale del progetto è di sviluppare e integrare reti di monitoraggio 4.0 e modelli probabilistici avanzati al fine di realizzare un approccio prognostico che consenta una gestione in sicurezza ed efficiente di componenti e sistemi, strutture ed infrastrutture, riducendo i costi operativi grazie alla eliminazione di azioni di manutenzione non necessarie o invasive.

Il carattere multidisciplinare dell'argomento ha richiesto l'integrazione di competenze proveniente da diverse aree metodologiche e tecnologiche, ed è per questo che il progetto ha coinvolto più università, alle quali sono stati affidati determinati obiettivi: Uniroma2, a cui è affidata la modellistica strutturale, l'Università di Messina per lo sviluppo di un sistema software e hardware per la realtà aumentata in ambito di previsione del degrado, il Politecnico di Milano per la prognostica, e infine l'Università di Bologna per la progettazione e sviluppo di reti di sensori multifunzione, per il processamento e gestione dati e per la validazione sperimentale.

Inoltre è in fase di realizzazione un campo prove situato a Bologna nel quale sono presenti strutture come serbatoi, tuberia, ponti in cemento armato e alluminio, su cui posizionare le reti di sensori al fine di testare il lavoro svolto.

Quindi l'obiettivo finale all'interno del quale si posiziona il mio progetto di tesi è quello di creare un'architettura Web of Things (WoT) per la manutenzione di componenti, strutture ed infrastrutture in ambienti di lavoro, grazie ad una rete di sensori IoT distribuiti e ad algoritmi innovativi per il processamento dei dati.

5.2 Obiettivi

Lo scopo del lavoro di tesi è la progettazione e realizzazione di una piattaforma che consente la gestione di dati IoT. Il sistema deve permettere la visualizzazione tramite grafici di serie di dati temporali, nonchè di effettuare queries direttamente su database, dando in seguito la possibilità di creare nuovi grafici e applicare tecniche di filtraggio personalizzabili. Tale sistema, come accennato nella sezione precedente, si posiziona all'interno del progetto Mac4Pro, e per tale motivo il tool realizzato è un componente che verrà integrato nell'architettura progettuale complessiva.

Il sistema è finalizzato all'interpretazione e all'analisi dei dati provenienti dalle reti di sensori posizionate sulle strutture del campo prova descritto in precedenza.
Per questo la piattaforma deve permettere l'integrazione di dati provenienti da una molteplicità di cluster differenziati a seconda della struttura monitorata. L'obiettivo è fornire all'utente uno strumento con il quale possa monitorare in tempo reale qualsiasi tipo di struttura, indipendentemente dalla tipologia di rete di sensori utilizzata, così che riesca ad analizzare e a identificare nell'immediato anomalie su qualsiasi tipo di dato. È fondamentale che l'interfaccia proposta abbia una complessità ridotta, in modo tale che qualsiasi utente possa utilizzare l'applicazione senza la necessità di avere competenze tecniche elevate.

La sua impostazione, vista l'architettura scalabile, lo rende utilizzabile per qualsiasi risorsa dati, in quanto il sistema, aderendo all'architettura WoT vista nel capitolo 3, è indipendente dal tipo di database utilizzato, ma richiede solamente che vi sia una connessione ad una Servient adibita alla persistenza dei dati.

5.3 Requisiti

5.3.1 Requisiti funzionali

Il sistema deve fornire le seguenti funzionalità principali:

- **Visualizzazione strutture connesse**
  L'utente deve potere visualizzare i dettagli delle strutture monitorate: devono essere riportate informazioni relative allo stato e alla composizione in termini di sensori e Thing di ciascun cluster di dati che identifica una struttura.

- **Visualizzazione di grafici**
  L'utente deve poter visualizzare attraverso grafici i dati in serie temporale provenienti dai sensori delle varie strutture monitorate.

- **Queries**
  L'utente deve essere in grado di eseguire una ricerca sui dati. Ciò deve avvenire attraverso delle queries specifiche al database in cui attraverso un form può richiedere i dati dei vari sensori. Vi deve essere la possibilità di poter aggregare dati, in modo tale che l'utente abbia piena libertà di selezionare una molteplicità di sensori e Thing relativi anche a strutture differenti. Inoltre tale ricerca può essere filtrata ulteriormente in base all'intervallo temporale di produzione dei dati che si desidera visualizzare.

- **Applicazione di filtri su queries**
  Il sistema, in fase di esecuzione di queries, deve dare la possibilità di applicare filtri sulla queries eseguita, ad esempio calcolare la media dei valori richiesti.

- **Creazione di un grafico**
  L'utente a seconda delle sue necessità deve avere la possibilità di comporre un grafico con i dati di suo interesse. Tale funzionalità sarà strettamente correlata alla funzionalità di queries descritta sopra, in quanto la composizione del grafico dovrà avvenire successivamente alla ricerca e selezione dei dati che l'utente vuole visualizzare.

- **Creazione di dashboard personalizzabili**
  Oltre alle dashboard predefinite presenti nel sistema, le quali sono suddivise per struttura monitorata, l'utente deve avere la possibilità di creare nuove dashboard personalizzabili. Tale creazione deve prevedere l'inserimento di un nome da assegnare alla nuova dashboard; all'interno di quest'ultima l'utente potrà creare nuovi grafici attraverso le funzionalità descritte in precedenza.

- **Visualizzazione real-time**
  Ciascun grafico presente nel sistema deve poter essere visualizzato in real-time: ciò significa che la serie dati temporali dovrà aggiornarsi in live con gli ultimi dati emessi dai sensori. Tale funzionalità dovrà essere opzionale, infatti l'utente potrà avviare e mettere in pausa la visualizzazione real-time a seconda dei propri interessi, qualora ad esempio voglia analizzare i dati di uno specifico intervallo temporale

- **Download dati**
  Il sistema deve permettere il download tramite browser delle risorse di dati presenti nel database. Tale funzione è dipendente dalla funzionalità di queries, in quanto solo dopo aver eseguito la ricerca dati tramite l'apposito form l'utente potrà scaricare i dati richiesti. Il formato del file predefinito è il CSV (Comma-separated values), un formato comune utilizzato per l'importazione e l'esportazione di tabelle di dati, ampiamente supportato.

- **Filtraggio temporale**
  All'interno di ciascuna dashboard dovrà essere presente un filtraggio temporale globale. Ciò significa che attraverso la scelta dell'intervallo di data/ora selezionato dall'utente tutti i grafici presenti all'interno di una dashboard dovranno aggiornare la serie di dati all'intervallo temporale stabilito.

- **Settaggio tag thing**
  Il sistema deve poter permettere all'utente di settare un tag all'interno di una Thing al fine di etichettare le rilevazioni provenienti dal sensore preso in considerazione, ad esempio per effettuare un test ed definirlo come tale.

- **Settaggio frequenza di aggiornamento**
  Il sistema deve poter permettere all'utente di settare un la frequenza con la quale vengono campionati e salvati i dati nel databes. Ciò deve

avvenure mediante la modifica di una proprietà all'interno del TD della Thing.

5.3.2 Requisiti tecnici

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

- **Supporto**
  La piattaforma deve essere accessibile tramite Web perciò dovrà essere compatibile con le ultime versioni dei principali browser. Affinchè sia soddisfatta la visualizzazione real-time, il sistema necessita di una connessione Internet stabile. Il deploy del sistema dovrà avvenire su un server fisico o su un cointainer su cloud, il cui indirizzo dovrà essere esposto e raggiungibile dall'utente

- **Architettura**
  L'architettura deve seguire i principi SOLID, pertanto deve essere facilmente estendibile, modulare e flessibile, sfruttando un'implementazione basata su interfacce ed API. In termini di estendibilità, l'architettura deve permettere di consumare qualsiasi tipo di Thing così da poter aggiungere nel tempo nuove funzionalità.

- **Usabilità**
  L'interfaccia deve presentare una struttura che sia utilizzabile da qualsiasi tipologia di utente, in modo tale che riesca a raggiungere i propri obiettivi derivanti dall'uso della piattaforma con facilità e immediatezza. Non è richiesto all'utente di configurare nè la rete di sensori nè il database da utilizzare. Tali funzionalità sono intrinseche nell'architettura del progetto complessivo.

5.4 Architettura

In questa sezione viene analizzata l'architettura della piattaforma di gestione e visualizzazione dati per Mac4Pro. Al fine di comprendere meglio la finalità del progetto svolto viene fornita anche una panoramica architetturale dell'intero progetto Mac4Pro, in quanto i vari componenti e le loro funzionalità sono strettamente dipendendenti tra loro.

L'architettura complessiva è raffigurata graficamente in figura 5.1, mentre in figura 5.2 è possibile osservare l'architettura nel dettaglio del Visualization.

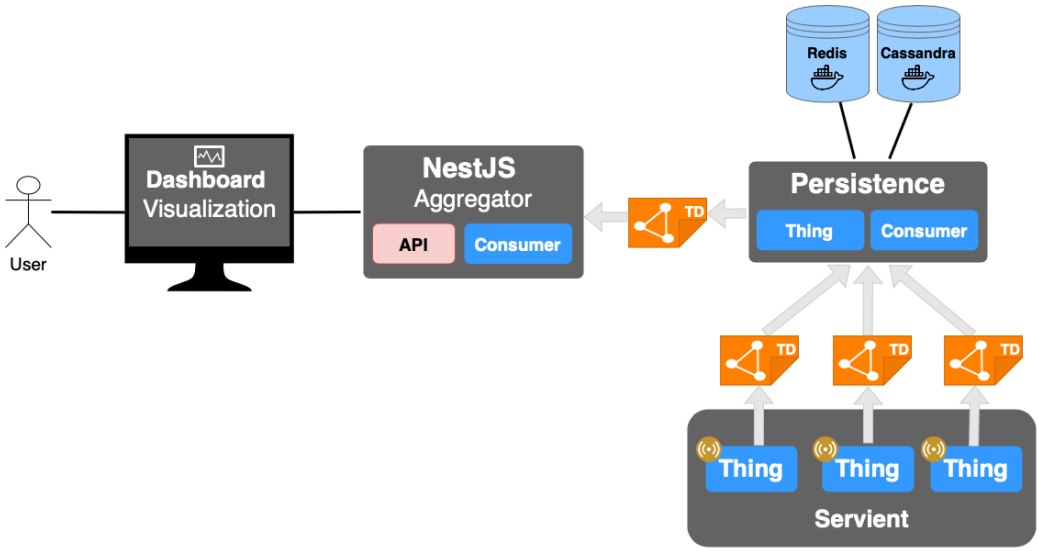

Figura 5.1: Architettura di Mac4Pro

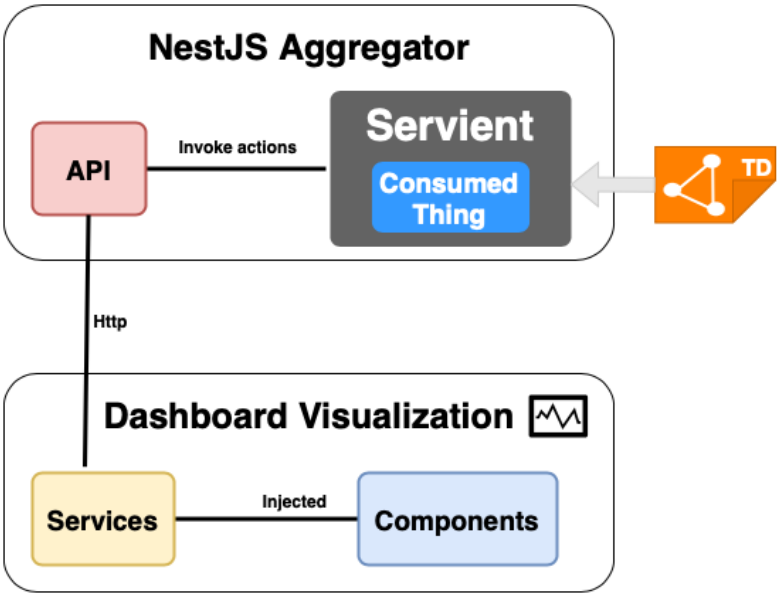

Figura 5.2: Architettura di Mac4Pro Visualization

5.4.1 Componenti

Il gruppo di componenti relativo al mio progetto di tesi è costituito dall'insieme degli elementi utili all'utente per interagire con l'intero sistema:

- **Dashboard Visualization**
  La Dashboard Visualization costituisce il lato client dell'applicazione web. Rappresenta l'interfaccia di monitoraggio del progetto Mac4Pro. Al suo interno sono presenti tutte le funzionalità descritte nella sezione 5.3.1. Attraverso l'interfaccia grafica della dashboard l'utente può interagire direttamente con il sistema sottostante, al fine di visualizzare tutti i componenti connessi, tra cui i clusters di sensori, esposti mediante gruppi di Thing, e le risorse dati prodotte dai sensori le quali sono memorizzate nel database distribuito. La Dashboard presenta due componenti principali: i Services e i Components. I services sono oggetti di tipo singleton che hanno l'obiettivo di organizzare e condividere logica dati e modelli tra i componenti. All'interno di uno dei services

dell'applicazione è implementata la logica di comunicazione con il NestJS Aggregator, che costituisce la parte server-side dell'applicazione. La comunicazione avviene tramite il protocollo HTTP, sfruttando le API messe a disposizione da NestJS Aggregator sulla base delle quali fornisce tutte le funzionalità previste. Invece i components sono delle entità configurabili e personalizzabili che a sua volta possono essere inseriti all'interno di altri componenti. Permettono di definire l'interfaccia grafica dell'applicazione. Al loro interno i componenti possono istanziare un service sfruttando il design pattern del Dependency Injection; in questo modo è possibile visualizzare i dati provenienti dalle API di NestJS Aggregator all'interno di un componente grafico.

- **NestJS Aggregator**
  NestJS Aggregator è la parte server-side dell'applicazione. Presenta una duplice comunicazione: da un lato, espone le API per la Dashboard per soddisfarne le richieste; dall'altro agisce come Consumer della Thing "Persistence". Ciò è dovuto al fatto che NestJS non è strutturato come una tradizionale applicazione server-side in cui è sempre presente una comunicazione diretta con un database associato. In questo caso è stato scelto di separare la logica di persistenza dei dati e affidarla ad una Servient, ovvero il componente Persistence. Quindi il compito di NestJS Aggregator è di consumare la Thing esposta dalla Servient Persistence al fine di poter invocare le azioni associate, dichiarate nella Thing Description. Perciò, ad ogni richiesta da parte della Dashboard eseguita mediante chiamate http alle API esposte da NestJS, quest'ultimo consuma la Thing di persistenza, per poi invocare l'azione necessaria a soddisfare la richiesta della Dashboard.
  Il nome NestJS Aggregator deriva dalla composizione di due elementi: NestJS è il framework utilizzato per sviluppare l'applicazione, mentre Aggregator indica il concetto per il quale è stato pensato questo componente, ovvero un aggregatore di dati che permetta di raccogliere dati da una molteplicità di risorse, combinarli e presentarli in un formato

riepilogativo al fine di eseguire analisi sui dati.

Ora verranno analizzate le funzionalità dei restanti componenti che compongono l'architettura del progetto Mac4Pro:

- **Persistence**
  Il persistence è il componente dell'architettura che si occupa della persistenza dei dati: ottiene i dati prodotti dai sensori e li memorizza all'interno del database. Tale componente si comporta sia come Servient che come Consumer, infatti consuma tutte le Thing presenti nelle varie strutture monitorizzate. Ottenute tali Thing, il persistence salva i valori contenuti all'interno di un database; in particolare è stato utilizzato Apache Cassandra[8], il cui deploy è distibuito su Docker-compose[9]. Inoltre è presente un meccanismo di caching implementato tramite Redis[10] per ridurre i tempi di risposta delle richieste al db Cassandra.
  Allo stesso tempo il persistence è esponene una Thing tramite TD utile al NestJS Aggregator per interagire con il database. La Thing Description della Servient Persistence presenta le seguenti actions:

  - *setUpdateFrequency*: setta il valore della proprietà updateFrequency, per aggiornare la frequenza di campionamento.
  - *setTagTest*: setta il valore della proprietà tagTest
  - *executeQuery*: permette di eseguire una query su Cassandra

  Il database è suddiviso in keyspaces, in cui ciascuno di essi indica la replica dei dati sui nodi. Per convenzione, è stato creato un keyspace per ciascuna struttura monitorata, il cui insieme è stato definito per convenzione con il nome di cluster. All'interno di tali cluster le tabelle sono suddivise per tipo di sensore.

[8] `http://cassandra.apache.org`
[9] `https://docs.docker.com/compose/`
[10] `https://redis.io`

Ogni tabella presenta i seguenti campi:

- Id (univoco)
- Timestamp
- tagTest
- Thing name
- Un campo per ogni tipo di valore emesso dal sensore (eg. x, y, z).

- **Servient e Network di Thing sensori**
  L'ultimo componente dell'architettura è costituito dall'insieme di Thing con cui vengono identificati i sensori presenti all'interno delle varie strutture monitorizzate. Ciascuna Thing può raggruppare più sensori, l'insieme di tutte le Thing viene esposta al Persistence attraverso una Servient. Quest'ultima contiene il collegamento a tutte le Thing sensore che verranno consumate dal Persistence per ottenere il Thing Descriptor contenente i metadati.

5.4.2 Flussi

Al fine di comprendere al meglio il ruolo di ogni componente, di seguito vengono descritti nel dettaglio alcuni dei flussi delle funzionalità principali dell'applicazione:

- **Esecuzione di una Query**

  1. L'utente accede alla pagina dedicata all'esecuzione di una Query tramite la barra laterale.
  2. In fase di inizializzazione della pagina, la Dashboard chiama tramite chiamata http GET l'API del NestJS Aggregator per ottenere i metadati di tutti i cluster, sensori e thing presenti all'interno del database, al fine di riempire il form di selezione.

3. Il NestJS Aggregator alla richiesta della Dashboard consuma la Thing di persistenza, invocando successivamente l'azione di 'execution query' sulla Persistence consumata, richiedendo tutto lo schema del db.
4. Il Persistence risponde all'azione invocata ritornando una struttura JSON contenente il metadata richiesto.
5. Il NestJS Aggregator, dopo aver mappato la struttura JSON in un oggetto, ritorna quest'ultimo alla Dashboard.
6. La Dashboard a partire dall'oggetto ritornato, riempie il form di multi selezione con gli elementi contenuti nell'oggetto (cluster, sensori e thing).
7. I punti precedenti fanno riferimento alla fase di inizializzazione della pagina; l'utente quando accede alla pagina avrà a disposizione un form in cui può selezionare i cluster (ovvero la divisione delle strutture), i vari sensori per ogni cluster e le thing che idenificano i sensori.
8. L'utente compone la query a proprio piacimento selezionando gli elementi dal form. Può selezionare più elementi per ciascun gruppo. Deve selezionare obbligatoriamente almeno un cluster e l'intervallo temporale. Al termine clicca sul button Execute query per eseguirla.
9. Il form di creazione query, a seguito del click sul button, controlla se i campi sono stati compilati correttamente, e in tal caso invia l'oggetto con gli elementi selezionati al componente che si occupa della creazione del grafico, che ritornerà un grafico con i dati dei sensori prescelti.

- **Download CSV**

  1. L'azione di Download dei dati in formato CSV si sussegue al flusso di esecuzione di una query.
  2. L'utente una volta completata la creazione della query, cliccando sul button Execute Query crea l'oggetto Query con gli elementi richiesti.
  3. La Dashboard invia l'oggetto Query a NestJS Aggregator tramite chiamata http POST.
  4. NestJS Aggregator a ricezione dell'oggetto Query controlla gli elementi presenti al suo interno, seguendo la logica già definita in precedenza nel flusso di creazione del grafico (punto 5).
  5. NestJS ritorna alla dashboard una lista di entry corrispondenti a quelle presenti nelle tabelle del database. Per ogni entri sono state aggiunte informazioni non presenti, quali il nome del cluster e del sensore corrispondente, in quanto tali informazioni erano omesse poichè corrispondono al nome del keyspace e della tabella.
  6. La dashboard attraverso una funzione all'interno di un service converte la lista ritornata in formato CSV, rendendola disponibile al download.
  7. L'utente può scaricare i dati in formato CSV cliccando sul button "Download CSV".

- **Creazione di dashboard personalizzabili**

  1. Inizialmente l'utente può trovarsi in qualunque pagina dell'applicazione, in quanto la creazione di una nuova dashboard avviene mediate il button presente nella barra laterale.
  2. Cliccando sul button in questione appare in primo piano una modal in cui l'utente deve inserire il nome della dashboard.

3. Il sistema richiede l'univocità dei nomi delle dashboard, controllando quindi la presenza di una dashboard già esistente con il nome inserito.
4. In caso di successo, l'utente si ritrova nella pagina della nuova dashboard appena creata.
5. L'utente ora può inserire nuovi grafici al suo interno tramite l'apposito form

- **Creazione di un grafico**

1. L'azione di creazione di un grafico avviene dopo il flusso di esecuzione di una query.
2. L'utente una volta completata la creazione della query, cliccando sul button Execute Query crea l'oggetto Query con i gli elementi richiesti.
3. La Dashboard richiama il componente di creazione grafico dandogli in input l'oggetto Query.
4. Il componente del grafico esegue una chiamata http POST a NestJS Aggregator inviando l'oggetto Query.
5. NestJS Aggregator a ricezione dell'oggetto Query controlla gli elementi presenti al suo interno, suddividendo la funzione in 3 casistiche:
   - Se l'utente ha selezionato solo il/i cluster, l'API consuma la Thing Persistence, da cui ottiene inizialmente i metadati della struttura database, per poi invocare l'azione di 'execution query' per tutte le tabelle presenti all'interno del cluster selezionato.
   - Se l'utente oltre ai cluster seleziona anche i sensori, l'API consuma la Thing Persistence e invoca l'azione di 'execu-

tion query' per tutti i sensori presenti all'interno del cluster selezionato, ritornando quindi l'intera tabella.

- Se l'utente oltre ai cluster e sensori seleziona anche le relative Thing, la funzione dell'API consuma la Thing Persistence e invoca l'azione di 'execution query' per le tabelle del cluster selezionato che identificano i sensori richiesti, filtrando le entry solamente a quelle relative alle Thing selezionate.

6. NestJS Aggregator ritorna alla Dashboard un oggetto key-value strutturato gerarchimente (Cluster $\rightarrow$ Sensor $\rightarrow$ Thing) in cui in posizione Thing sono presenti tutte le entry della tabella sensore relativa alla specifica Thing.
7. L'oggetto ritornato viene gestito dal componente della Dashboard designato alla creazione del grafico. Per ogni valore del sensore (es. x, y, z) relativo ad una specifica Thing viene generata una serie. Tale serie verrà visualizzata con un grafico a linea e identificabile in legenda con la seguente convenzione:
   NomeSensore_NomeThing_Y.

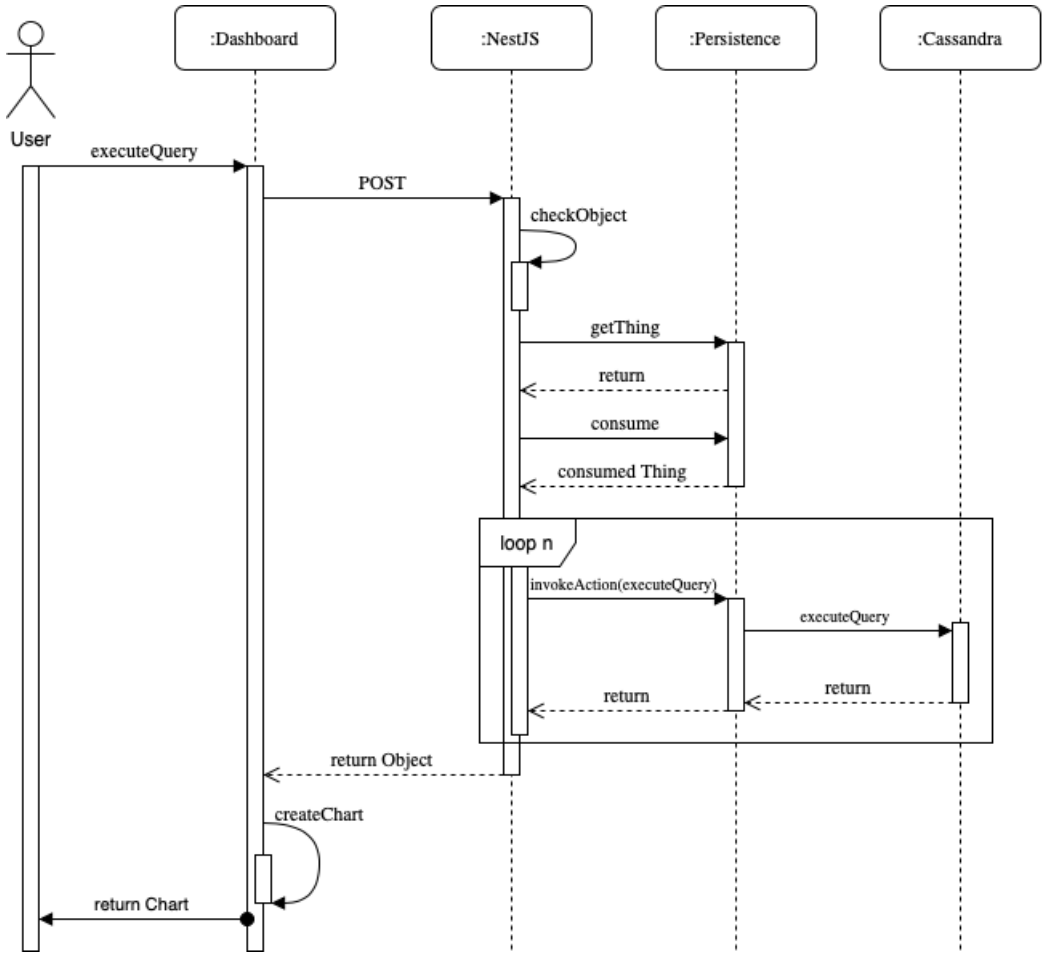

Figura 5.3: Sequenza di azioni invocate durante l'operazione di creazione di un grafico

Capitolo 6

Implementazione

Lo sviluppo del sistema ha visto coinvolto l'integrazione di più componenti al fine di fornire le funzionalità previste descritte in precedenza. All'interno di questo capitolo vengono analizzate le tecnologie utilizzate e i vari moduli realizzati, focalizzandosi sui problemi più significativi riscontrati e sulle scelte progettuali adottate.

6.1 Tecnologie

Essendo praticamente un progetto "full-stack" sono state coinvolte più tecnologie: in questa sezione verranno descritte nel dettaglio, soffermandomi sulle motivazione che mi hanno portato ad adottare queste scelte.

NestJS Aggregator rappresenta il lato server-side dell'applicazione, il quale è stato sviluppato in Node.js v10.x mediante il framework NestJS. Per funzionare correttamente necessita delle seguenti dipendenze:

- **@nestjs**: è il package principale, ovvero il node module su cui si basa il framework NestJS. Al suo interno sono presenti i moduli base dell'architettura necessari per il funzionamento dell'infrastruttura.

- **@node-wot**: è il package necessario per implementare la Servient in Node.js al fine di Consumare la Thing persistence utile ad invocare azioni sul database ed ottenere le risorse dati

Il lato client, ovvero la Dashboard Visualization, è sviluppata in Angular v9.x ed utilizza una molteplicità di package. Oltre ai package di default presenti in angular, ne sono stati utilizzati ulteriori, tra cui: **@ngx-bootstrap**, per avere componenti pronti, estendibili ed adattabili in stile bootstrap; **@ngx-widget-grid**, per creare una griglia i cui componenti all'interno sono spostabili e ridimensionabili; **@ngx-echarts** per la creazione di grafici aggiornabili in tempo reale; **angular2-multiselect-dropdown** per la creazione di form a scelta multipla.

6.1.1 NestJS

NestJS è un framework open-source per la creazione di applicazioni server-side in Node.js altamente efficienti e scalabili. Fornisce alle applicazioni back-end una struttura modulare per l'organizzazione del codice in moduli separati. L'organizzazione della sua struttura a moduli è dovuto al fatto che è fortemente ispirato ad Angular[11]. Inoltre è scritto in Typescript, permettendo quindi di scrivere Javascript tipizzato assicurando una certa consistenza sintattica.
Al suo interno utilizza Express come framework per l'http server. Nest offre un livello di astrazione al di sopra di questo framework (Express), ma permette anche di esporre direttamente le API allo sviluppatore, consentendo di utilizzare una miriade di moduli di terze parti.

Si è scelto di utilizzare NestJS in quanto fornisce un'architettura applicativa pronta all'uso e l'applicazione creata, oltre ad essere scalabile e liberamente accoppiabile è anche facilmente gestibile. Un'altro motivo che mi ha spinto a scegliere NestJS è la sua architettura ispirata ad Angular: in questo

[11] `https://angular.io`

modo ho potuto utilizzare lo stesso linguaggio (Typescript) sia per la parte front-end che back-end, inoltre condividendo la stessa filosofia modulare passare dal back-end (Nest) al front-end (Angular) è stato come lavorare nello stesso progetto, riducendo i tempi nel cambio del contesto.
I componenti fondamentali (building blocks) di un applicazione NestJS sono:

- **Controller**
  È responsabile della gestione di eventuali richieste in arrivo e della restituzione di risposte lato client. Ad esempio, se si effettua una chiamata API ad un particolare endpoint, come "/persistence/executeQuery", il controller riceverà questa richiesta e in base alle risorse disponibili, restituirà la risposta appropriata. Nest.js è stato strutturato in modo tale che il meccanismo di routing sia in grado di controllare quale controller sarà responsabile della gestione di una particolare richiesta.

- **Services**
  I services, chiamati anche providers, come in Angular sono progettati per astrarre qualsiasi forma di complessità e logica. Un services in Nest.js è una classe Typescript con un decoratore speciale @Injectable nella parte superiore.

- **Moduli**
  I moduli consentono di raggruppare i file correlati. Sono file in Typescript decorati con il @Module decorator, il quale fornisce dei metadati che Nest utilizza per organizzare la struttura dell'applicazione. Ogni applicazione NestJS deve avere almeno un modulo, generalmente indicato come modulo root. All'interno della mia applicazione, come vedremo, oltre al modulo root sono presenti due ulteriori moduli ciascuno per determinate funzionalità.

Nest dispone di un command line tool (nestjs/cli) che permette di generare con estrema facilità i componenti appena descritti nonchè visualizzare informazioni sul progetto.

6.1.2 Angular

Angular [12] è un framework open-source per la creazione di applicazioni Web single-page in HTML e Typescript. È scritto in Typescript, e sviluppato principalmente da Google; è l'evoluzione del suo predecessore AngularJS. Angular fornisce un' architettura strutturata che garantisce il massimo livello di riutilizzabilità e modularizzabilità del codice, caratteristiche molto importanti quando si sviluppano applicazioni web di medio-alta complessità, nonchè uno dei principali motivi per cui ho scelto di adottarlo per la parte front-end.
L'architettura di Angular è basta sugli *NgModules* che forniscono un contesto di compilazione per i componenti. All'interno dell'applicazione sono presenti più NgModules: è sempre presente un modulo root per le istruzioni del caricamento iniziale, dopo di che possono essere presenti più moduli funzione.

I componenti definiscono le views, ovvero l'insieme di elementi grafici che Angular permette di utilizzare e modificare a seconda della logica del programma. Inoltre i componenti utilizzano i services, che forniscono funzionalità non strettamente connesse alle views. I services, come in Nest, possono essere iniettati nei componenti come dipendenze, rendendo il codice modulare, riutilizzabile ed efficiente.

Sia i componenti che i servizi sono semplicemente delle classi, con decoratori che ne definiscono il tipo e forniscono metadati che indicano ad Angular come usarli. I metadati per una classe di componenti lo associano a un modello che definisce una vista. Un modello combina HTML ordinario con direttive Angular e markup di binding, che consentono ad Angular di modificare l'HTML prima di del rendering per la visualizzazione. I metadati per una classe service forniscono le informazioni necessarie ad Angular per renderli disponibili ai componenti attraverso la Dependency Injection (DI). La DI è un design pattern che semplifica altamente l'applicazione rendendo

[12] `https://angular.io`

i componenti più riutilizzabili, più facili da gestire e testabili. Ho potuto testare l'utilità di questo pattern durante l'utilizzo del Service adibito alla comunicazione con NestJS: il Service presenta le implementazioni delle varie chiamate http alle API di NestJS, e per visualizzare l'oggetto della risposta (es. dati dei sensori) all'interno di un componente grafico è sufficiente dichiare il Service nel costruttore della classe del componente, così da richiamarlo successivamente al fine di sfruttarne le funzionalità per la comunicazione con NestJS. In questo modo componenti diversi possono utilizzare lo stesso Service al loro interno.

Oltre al vantaggio di un'architettura component-based, la scelta di adottare Angular come framework per il lato front-end del progetto deriva dal fatto che la sua struttura modulare ed estendibile rendono lo sviluppo altamente efficiente; questo tipo di architettura però si è rivelata inizialmente ostica in quanto nelle precedenti esperienze di sviluppo di un' applicazione front-end non avevo mai adottato un'architettura modulare, bensì semplici file HTML con script javascript annessi. Inoltre il fatto di poter utilizzare Typescript, che è il super-set di Javascript tipizzato, ha apportato una maggior sicurezza e servizi di navigazione, nonchè un autocompletamento e un refactoring del codice migliore rispetto a quelli offerti da Javascript. Infine anche l'architettura MVC ha permesso una notevole riduzione di codice, poichè non richiede linee di codici complicate per implementare i vari modelli, facendo si che vi sia una miglior comprensione di come funzioni l'applicazione ancor prima dell'esecuzione.

6.1.3 Docker

Docker[13] è uno strumento che permette di pacchettizzare un'applicazione e le sue dipendenze all'interno di un container virtuale che può essere eseguito su qualsiasi server Linux. L'intera piattaforma realizzata (Visualization e NestJS Aggregator, nonchè il Persistence e il database) sono stati preparati

[13] `https://www.docker.com`

per un deploy su un cluster Docker Compose come Stack, ovvero una serie di Service interconnessi tra loro. In questo modo si ha un triplice vantaggio: un sistema altamente scalabile, replicabile su più cluster, e una gestione rapida dell'infrastruttura.

6.1.4 Eclipse Thingweb node-wot

Nella realizzazione del sistema, al fine di poter consumare la Thing del componente Persistence, è necessaria un'implementazione di una Servient all'interno di NestJS. Attualmente l'unica supportata e garantita dal W3C [18] è quella sviluppata dalla Eclipse Foundation: Eclipse Thingweb node-wot[14]. Si tratta di una Servient sviluppata per Node.js v.10.x che richiede alcune dipendenze aggiuntive, quali Python 2.7, make e un compilatore C/C++, come ad esempio GCC.

6.2 Moduli

6.2.1 NestJS Aggregator

Il server NestJS è suddiviso in due moduli principali: Dashboard-api e Servient-persistence.

Dashboard-api

Questo modulo è designato alla gestione delle dashboard, ovvero alla creazione e al salvataggio delle dashboard di default e quelle personalizzabili create dall'utente. La persistenza delle configurazioni avviene all'interno di un file *JSON*, composto da più oggetti "dashboard", ovvero liste di query eseguite e salvate dall'utente che identificano i grafici. Il modulo è composto da un controller e da un service. Nel controller sono presenti le API esposte al client, tra cui abbiamo i metodi di *get* e *set* di una Dashboard, con cui vengono ottenute le dashboard esistenti e aggiunte delle nuove. All'interno

[14] `https://github.com/eclipse/thingweb.node-wot`

del service è implementata la logica con viene aggiunta una nuova dashboard e la funzione con la quale si ottengono i grafici creati dall'utente.

Servient-persistence

Questo modulo costituisce il cuore del NestJS Aggregator, in quanto contiene tutte le funzionalità principali per comunicare con il Persistence. Anch'esso è composto da un controller e da un service. All'interno del service viene utilizzato il modulo node-wot al fine di instanziare una Servient e consumare la Thing di persistenza. Ciò avviene mediante due funzioni asincrone:

- *getThing*, la quale esegue la fetch dell'indirizzo della Thing dato in input, ritornando come risposte la Thing richiesta;
- *invokeActions*, in cui tramite la servient istanziata si consuma la Thing data in input tramite una chiamata asincrona, per poi invocare l'azione desiderata tramite il metodo invokeActions. Il metodo ritorna il risultato dell'azione invocata sulla Thing, che verrà gestito nel controller.

```
@Injectable()
export class ServientService {

  async getThing(title:string, host:string, port:string): Promise<JSON> {
    const data = await fetch('http://'+ host +':'+ port +'/'+ title);
    return data.json();
  }

  async invokeAction(action: string, thing: JSON, param: any) {
    const WoTServient = await servient.start();
    const actionReturn = WoTServient.consume(thing).then(
      async(consumedThing) => {
        return await consumedThing.invokeAction(action,param); });
    servient.shutdown();
    return actionReturn;
  }
}
```

Listing 6.1: Classe ServientService con le funzioni di getThing e invokeAction

Analizzando invece il file controller del servient-persistence, al suo interno sono presenti tutte le chiamate API esposte al client. Ad ogni chiamata API corrisponde una funzione al cui interno viene iniettato il ClusterService che permette di consumare la Thing persistence ed invocare le azioni esposte. Le API con le funzioni più articolate corrispondono alle richieste di query da parte del client e al download della query in formato CSV, in quanto all'interno di tali funzioni il controller deve analizzare l'oggetto ritornato dalla POST poichè a seconda dei parametri che l'utente ha scelto di visualizzare verrà richiamata un'ulteriore funzione specifica per il tipo di richiesta, che è addetta ad invocare l'azione di *execute query*, passando come parametri il cluster, i sensori e le thing richieste dall'utente.
Infatti all'interno del controller sono presenti molte API in base alle combinazioni delle richieste possibili. Ad esempio se l'utente ha scelto nel form di creazione query solamente i cluster, significa che dovranno essere ritornati all'utente tutti i sensori e le thing presenti al suo interno. Però la funzione API responsabile di ricevere l'oggetto Query dal client, non conoscendo la composizione del cluster e quali sensori possiede, inizialmente deve consumare la Thing persistence al fine di ottenere i metadati del database e analizzare le tabelle sensore contenute nel cluster selezionato. In seguito per ogni tabella verrà invocata l'action di 'execute query' che ritorna tutte le entry della tabella sensore.

Un'ulteriore funzionalità presente nel controller è la funzione di ordinamento temporale dei dati. Tale funzione è molto importante al fine della corretta visualizzazione grafica dei dati, poichè i dati ritornati dalle tabelle del database non sono ordinati temporalmente a causa dell'architettura distribuita sulla quale sono memorizzati. Tale funzione viene richiamata a seguito del ritorno dei dati da parte della Thing persistence consumata. Inoltre ulteriori funzionalità interne al Servient-persistence sono le actions di executeQuery e updateFrequency, in cui si vanno ad aggiornare in maniera diretta tali proprietà presenti nella Thing persistence.

Nel listato di codice seguente viene proposto un esempio di una chiamata API relativo al settaggio del tag test.

```
//TAG TEST
@Get('setTagTest/:tag')
async setTagTest(@Param('tag') tag) {
  const json = await this.servientService.getThing(
    this.configService.get<string>('db_servient.title'),
    this.configService.get<string>('db_servient.host'),
    this.configService.get<string>('db_servient.port')
  );

  return this.servientService.invokeAction('setTagTest',json,tag.toString())
}
```

Listing 6.2: Chiamata API del servient-persistence controller

6.2.2 Dashboard Visualization

L'architettura del client Dashboard Visualization è quella sicuramente più complessa. Per poter comprenderla meglio, riporto un'immagine che mostra la composizione dei vari moduli (Figura 6.1). In particolare, viene mostrata la struttura generale del progetto (src), per poi addentrarci nel dettaglio delle tre folder principali: **core**, al cui interno sono presenti i services, **dashboard**, che contiene tutte le views della dashboard e infine **shared**, al cui interno è stato scelto di mettere i componenti condivisi, ovvero quelli presenti in ogni view dell'applicazione.

Come visibile in figura 6.1 la struttura dell'applicazione front-end è altamente modulare. Tra i vari services contenuti nel modulo core, uno dei più importante è il cluster-service. Attraverso esso infatti l'applicazione comunica con NestJS Aggregator mediante le API esposte dal modulo servient-persistence. Il cluster-service è composto da una serie di funzioni asincrone responsabili di eseguire chiamate http alle API di NestJS. La chiamata può essere associata ad una GET o ad una POST.

Figura 6.1: Struttura dei moduli dell'applicazione in Angular

Il dashboard service, interno al modulo core, è destinato all'interazione con le API NestJS per la memorizzazione delle dashboard create dall'utente, quindi anche in questo caso sono presenti chiamate http REST. Infine sono presenti due servizi per la gestione del download del CSV e per la gestione della data globale con cui aggiornare tutti i grafici di una dashboard.
Un esempio di tali chiamate è visibile nel codice che segue.

```
1 public postQuery(query): Observable<any> {
2   const apiUrl = "${environment.apiUrl}/persistence/post/query";
3   return  this.http.post(apiUrl,query).pipe(catchError(ClusterService.
      handleError));
4 }
5
6 public getDbMetadata(): Observable<any> {
7   const apiUrl = "${environment.apiUrl}/persistence/get/db_metadata/";
8   return  this.http.get<any>(apiUrl).pipe(catchError(ClusterService.
      handleError));
9 }
```

Listing 6.3: Chiamate http GET e POST alle API di NestJS

Analizzando la struttura del modulo dashboard in figura 6.1 possiamo osservare che al suo interno sono presenti tutti i moduli che contribuiscono alla renderizzazione delle view dell'applicazione. Quest'ultima si compone di tre pagine: *landing-page*, che costituisce la pagina iniziale all'apertura dell'applicazione, al cui interno viene utilizzato il modulo cluster-info; *query-page* in cui è possibile effettuare una query al database e visualizzarne il risultato grafico, che a sua volta utilizza i componenti query-form e fixed-chart; infine *dashboard-grid*, ovvero la pagina che identifica una qualsiasi dashboard.
Al suo interno sono presenti n istanze del componente fixed-chart, con n che identifica il numero di grafici creati dall'utente. Fixed-chart è uno dei moduli più corposi dell'applicazione poichè ha una duplice funzione: deve eseguire la chiamata POST della query all'API di NestJS e gestirne la visualizzazione in real-time. In fase di inizializzazione del componente fixed-chart viene passato in input un oggetto "query" contenente gli elementi da visualizzare scelti dall'utente. Il componente non fa altro che eseguire la subscribe sulla funzione postQuery vista in precedenza, passando come parametro l'oggetto "query". È poi compito del service iniettato nel componente eseguire la chiamata http all'API specifica e ritornare le risorse dati ottenute dal persistence. A questo punto vengono inizializzate le opzioni per la creazione del grafico e viene mappato l'oggetto ritornato dalla chiamata in un formato consono al grafico (secondo il formato richiesto dal package ngx-echarts) così da riprodurre la serie temporale del dato correttamente.

Capitolo 7

Validazione

In questo capitolo viene presentato lo scenario di validazione su cui verrà testato l'intero progetto Mac4Pro. Inoltre verranno analizzati i risultati ottenuti dal lavoro progettuale svolto al fine di valutare come tale architettura soddisfi i requisiti stabiliti inizialmente.

7.1 Scenario

Lo scenario in cui verrà valutata l'intera architettura progettuale all'interno della quale si inserisce il mio lavoro di tesi sarà il campo prove descritto in precedenza dedito a testare i sensori, l' infrastruttura di rete e l' architettura WoT su strutture fisiche.

In particolare il campo prove è istituito in due differenti stabili dell'Università di Bologna, ovvero il Laboratorio di Ingegneria Idraulica (LIDR) e il Laboratorio di Ingegneria Strutturale e Geotecnica (LISG). All'interno di quest'ultimo sono state poste differenti strutture di differenti materiali, come ponti e strutture metallici. Su di esse sono stati posti dei sensori, quali accelerometro e giroscopio. In figura 7.1 è possibile osservare uno scenario di quanto descritto istituito presso il LISG: è stata disposta una struttura metallica sulla quale sono stati posizionati tali sensori. Ognuno di essi dispone

di un Thing Descriptor che ne descrive le proprietà del sensore nonchè i valori emessi. Tutti i sensori di una data struttura sono connessi ad un gateway che rappresenta la struttura complessiva, come descritto nella sezione 5.4: esso raccoglie l'insieme di web thing e espone in rete il loro indirizzo.

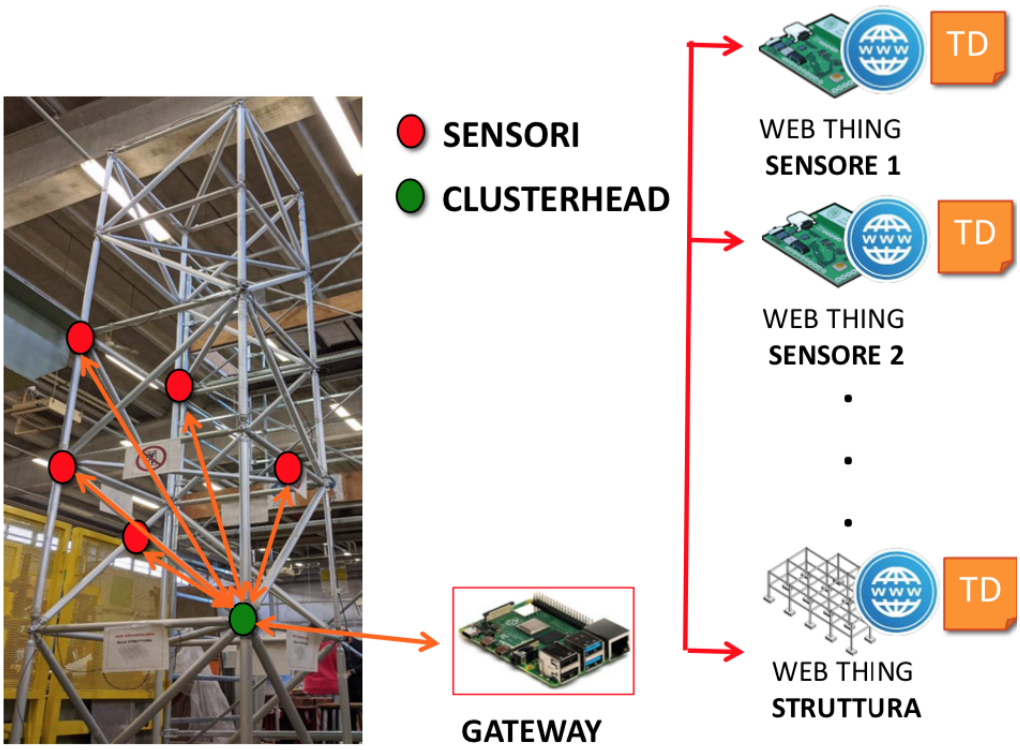

Figura 7.1: Scenario di validazione presso il LISG @UNIBO

Lo scopo di tale scenario è osservare come i valori emessi dai sensori subiscano variazioni nel momento in cui la struttura è oggetto di sollecitazioni. L'infrastruttura WoT creata permette di rilevare in tempo reale i cambiamenti, infatti dopo essere stati opportunamente salvati è possibile visualizzare le serie di dati raccolte attraverso i grafici della dashboard. Il passo successivo nonchè lo scopo di tutto il progetto Mac4Pro è eseguire l'analisi prognostica sui dati pervenuti per prevenire interventi di manutenzione. Infatti altre strutture destinate al monitoraggio sono ponti in calcestruzzo o alluminio, opere civili che negli ultimi anni sono stati oggetto di numerevoli discussioni

e problemi in termine di manutenzione e controllo dello stato di degrado. Per questo, come visibile in figura 7.2 all'interno del campo prove sono presenti anche queste tipologie di strutture per testare l'architettura progettata con differenti materiali poichè le sollecitazioni che si verificano su opere civili subiscono modifiche sia in base al materiale con cui sono fatte che in base alla dimensione della struttura. L'obiettivo finale è portare le metodologie e le tecnologie realizzate durante il progetto di ricerca in ambito industriale, personalizzando la rete di sensori in base alle esigenze dello scenario applicativo di interesse.

Figura 7.2: Scenario di strutture presenti nel campo prove @UNIBO

7.2 Casi d'uso

In questa sezione vengono analizzati i casi d'uso presenti nell'applicazione, ottenuti a seguito della realizzazione del lavoro di tesi descritto fino ad ora. L'obiettivo iniziale di fornire all'utente un'applicazione dalla quale poter monitorare e gestire i dati relativi ai sensori delle strutture descritte in precedenza è stato raggiunto secondo i requisiti stabiliti. Il risultato finale è quindi una dashboard visualizzabile tramite web browser connessa all'intera infrastruttura, tramite la quale l'utente ha pieno controllo delle strutture connesse e può sfruttare le funzionalità dell'applicazione per gestire tutta la mole di dati prodotta dai sensori e salvata all'interno del database.
Di seguito vengono descritti i casi d'uso che rappresentano le funzionalità dell'applicazione messe a disposizione all'utente.

Visualizzazione informazioni delle strutture connesse

Quando l'utente accede alla piattaforma ha subito una visione completa delle varie strutture online. Infatti, come visibile in figura 7.3 l'homepage dell'applicazione propone informazioni relative ai cluster connessi, con la composizione di sensori e relative thing.

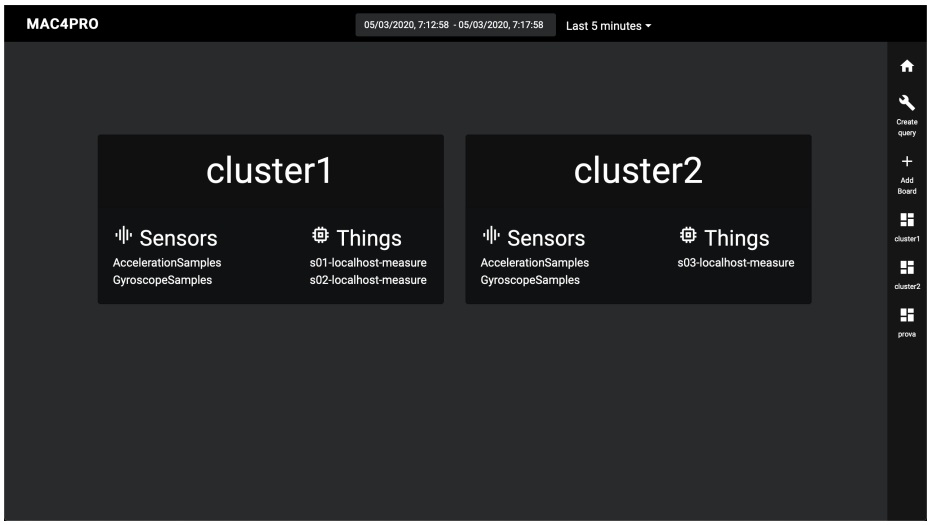

Figura 7.3: Homepage dell'applicazione

Visualizzazione della dashboard di un cluster

Nel lato destro della pagina (Figura 7.3) è presente un menu laterale con il quale l'utente può navigare tra le pagine dell'applicazione così che abbia sempre a disposizione nell'immediato tutte le dashboard e la funzionalità di query. Cliccando su una delle strutture mostrate nell'homepage o direttamente nell'icona laterale si potrà accedere alla dashboard relativa alla data struttura. Per ognuna delle varie strutture connesse, è stata dedicata una dashboard di default contenente grafici predefiniti suddivisi per tipo di sensore e thing (Figura 7.4).

Visualizzazione real-time di una serie di dati

I grafici, come già anticipato, dispongono della funzionalità live: può essere abilitata mediante il pulsante start e messa in pausa a seconda delle esigenze di visualizzazione (Figura 7.4). L'utente può attivare questa funzionalità sia su un grafico interno ad una delle pagine dashboard create, sia su un grafico ottenuto a seguito dell'esecuzione di una query nella pagina dedicata alla creazione di Query.

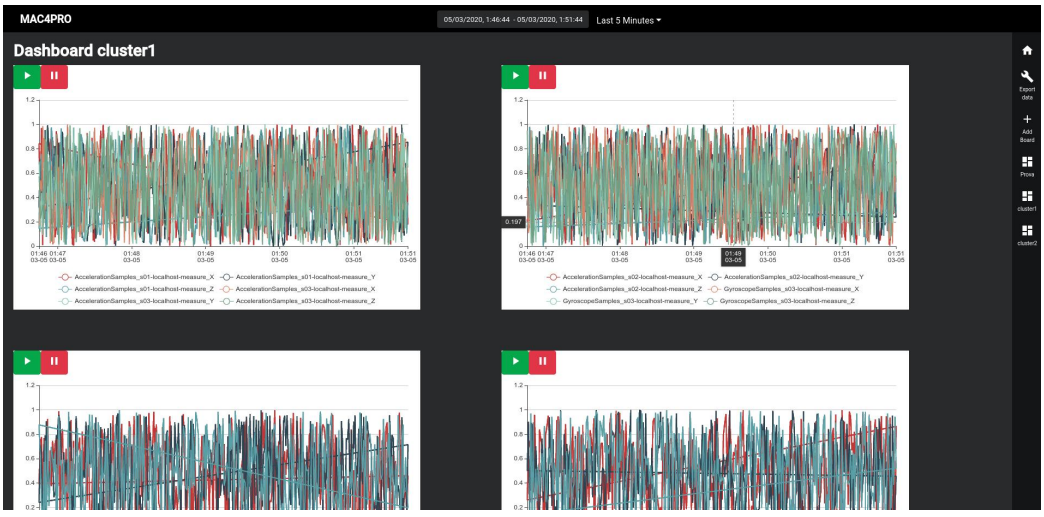

Figura 7.4: Pagina dashboard dell'applicazione

Modifica intervallo temporale dei grafici

Nella barra di navigazione posta in cima alla pagina è presente un date-picker: selezionando un intervallo temporale, che può variare di più giorni/-mesi, si modificherà l'intervallo di tempo per tutti i grafici contenuti nella pagina (Figura 7.5). Inoltre a fianco è presente un menu a tendina con degli shortcut per gli intervalli più recenti (ultimo minuto, ultimi 5 minuti, ultima ora, ecc.) al fine di velocizzare le scelte dell'utente nell'ottenere le ultime serie di dati. La scelta di rendere tali funzionalità globali e non interne a ciascun grafico deriva dal fatto che in un sistema di monitoraggio per dare coerenza ad un confronto tra più istanze è necessario che esse condividano lo stesso asse delle ascisse; inoltre tale scelta è in linea con gli strumenti di monitoraggio dati IoT analizzati nel capitolo 4.

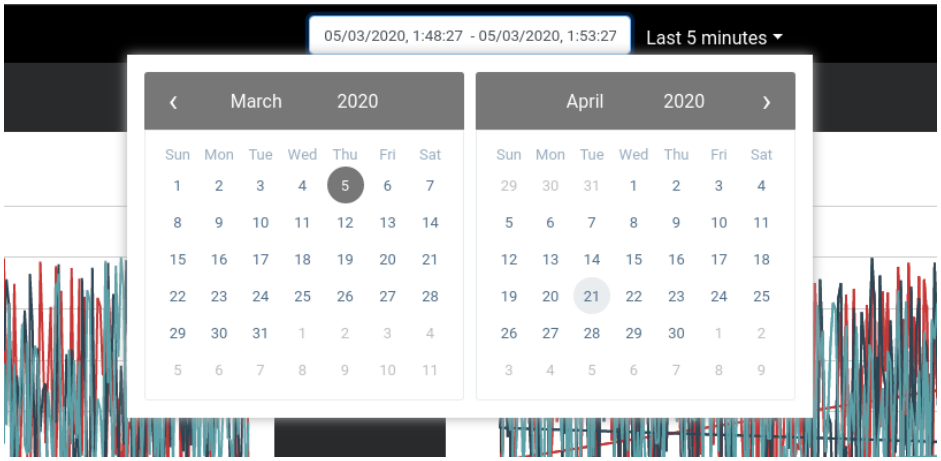

Figura 7.5: Strumento di selezione intervallo temporale

Creazione di una query

La funzionalità principale dell'applicazione nonchè uno dei requisiti fondamentali, è la creazione di query per interrogare il database ed estrapolare i dati dei sensori memorizzati.

Ciò avviene nella pagina "Create Query" accessibile dal menu laterale tramite l'apposito button.

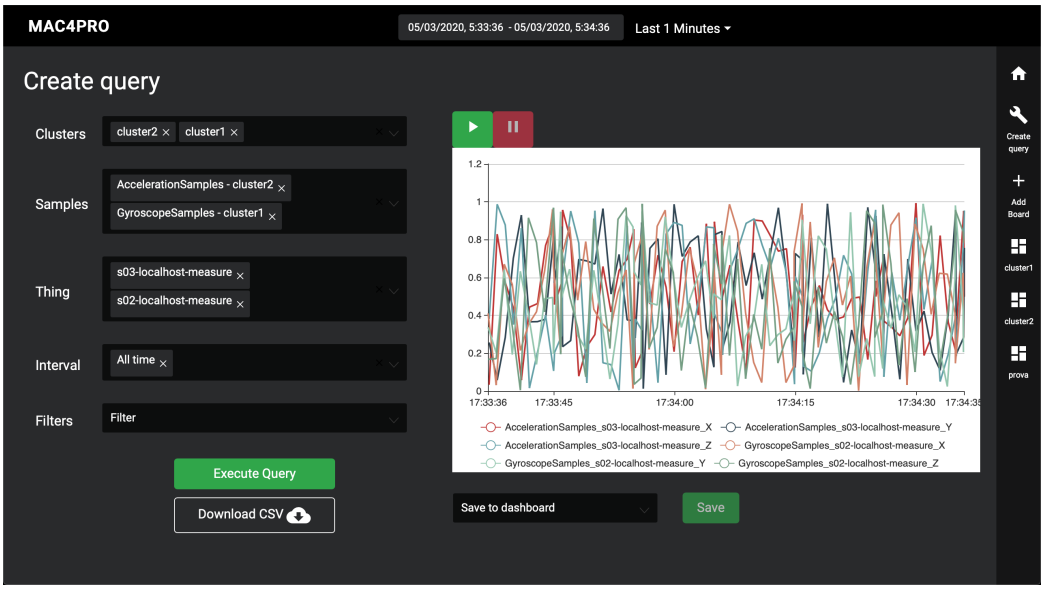

Figura 7.6: Pagina di creazione di una query

All'interno di questa sezione l'utente può comporre una query la cui complessità non ha limiti: infatti, come visibile in figura 7.6, il form permette di selezionare una molteplicità di istanze per ciascun campo presente. Il contenuto dei campi che è possibile selezionare si aggiorna in automatico in base ai campi superiori selezionati. Terminata la creazione, per rendere la query attiva basterà semplicemente cliccare sul button Execute.

La scelta di non porre limiti alla composizione della query, permettendo quindi la multi selezione, è dovuta al fatto che qualsiasi utente, in particolare gli specialisti del settore, potrebbe avere la necessità di mettere sullo stesso grafico serie di dati di sensori diversi o addirittura di strutture diverse per confrontarne l'andamento. Di conseguenza è compito dell'utilizzatore dell'applicazione generare un grafico che sia coerente e sensato. Gli unici vincoli posti all'utente per validare la query riguardano la selezione obbligatoria di almeno un cluster e dell'intervallo temporale.

Visualizzazione dati di uno specifico sensore

L'utente può visualizzare i dati di uno specifico accedendo alla pagina "Create query" come descritto nel caso d'uso specifico. Nel form presente all'interno della pagina, l'utente dovrà selezionare i cluster d'interesse e in seguito il sensore prescelto contenuto nel cluster. Inoltre può decidere di filtrare ulteriormente i dati dello specifico sensore selezionando solamente quelli provenienti da una data Thing. A creazione compiuta della query, a lato del form verrà disegnato il grafico contenente il sensore richiesto dall'utente come visibile in figura 7.7. Anche in questo caso è presente la modalità live che viene attivata a scelta dell'utente.

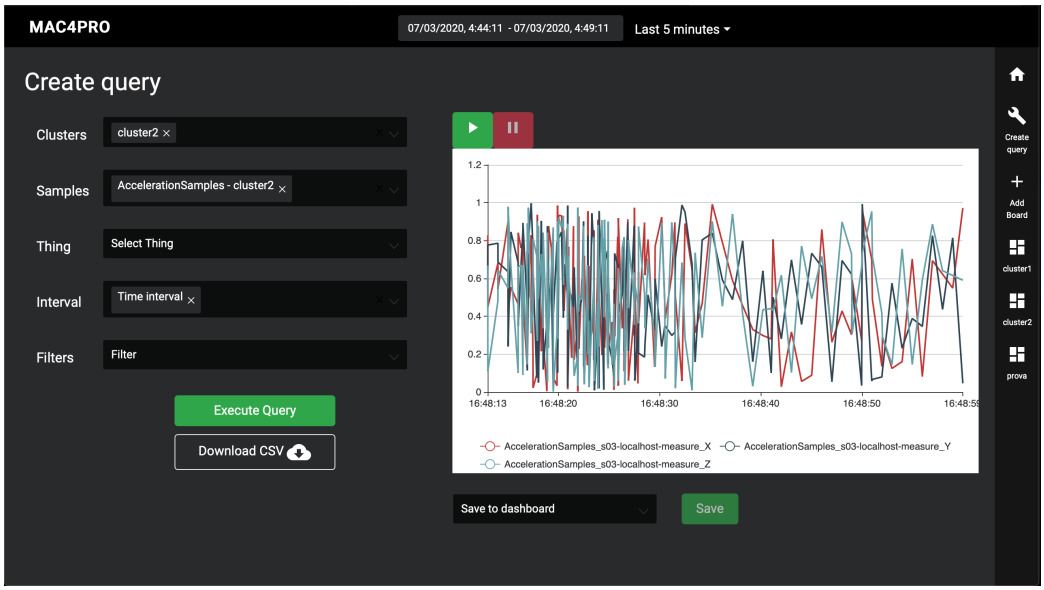

Figura 7.7: Visualizzazione dati di uno specifico sensore

Esportazione dati

L'utente può esportare i dati presenti nel database accedendo alla pagina "Create query" (Figura 7.6) come descritto nel caso d'uso specifico. Tramite l'apposito form l'utente seleziona i sensori di cui vuole esportare i dati, per poi eseguire la Query. L'esportazione sarà possibile quando il button "Download

CSV" risulterà attivo; in seguito attraverso un click su di esso sarà possibile eseguire il download dei dati in formato CSV.

Creazione di una nuova dashboard

L'utente può creare una nuova dashboard personalizzata nella quale salvare grafici. L'utente potrà eseguire l'azione di creazione di una dashboard accedendo tramite l'apposito button "New board" presente nella barra laterale. A seguito del click, apparirà una modal in cui verrà richiesto all'utente di inserire il nome della nuova dashboard (Figura 7.8). Per un corretto utilizzo del sistema, è necessario che le dashboard abbiano un nome univoco. Pertanto il sistema controllerà il nome inserito prima di poter procedere alla creazione. La nuova dashboard creata verrà aggiunta nella barra laterale tramite cui sarà accessibile.

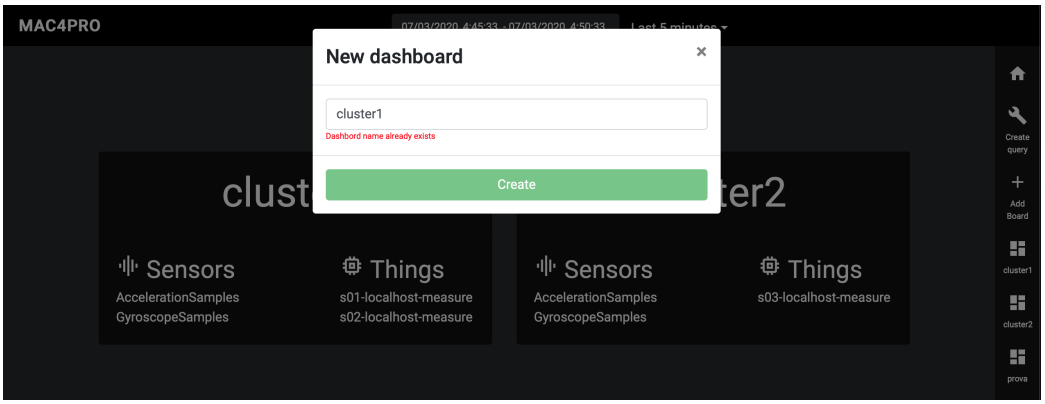

Figura 7.8: Modal di creazione di una nuova dashboard

Salvataggio di un grafico in una dashboard

L'utente, una volta creata la query e visualizzato il grafico, potrà scegliere se salvare il grafico creato all'interno di una delle dashboard presenti nel sistema, mediante la sezione di salvataggio posta sotto il grafico (Figura 7.7).

Capitolo 8

Conclusioni e sviluppi futuri

In questo elaborato è stata presentata la realizzazione di un'applicazione per la gestione di dati IoT per il monitoraggio strutturale secondo l'architettura WoT proposta dal W3C.

Nella prima fase è stato necessario uno studio preliminare degli aspetti tecnologici relativi alla situazione attuale in merito al mondo IoT e WoT, soffermandomi con particolare attenzione al contesto applicativo del monitoraggio strutturale. Successivamente ho approfondito nel dettaglio l'architettura WoT in quanto rappresenta il fulcro innovativo su cui si basa il progetto di ricerca Mac4Pro. Per avere una visione completa del progetto al quale ho contribuito con il mio lavoro di tesi è stato indispensabile analizzare anche le altre componentistiche dell'infrastruttura complessiva. Infatti alcune fasi di progettazione e sviluppo sono stati svolte in parallelo con il progetto Persistence affidato ad un collega del corso di laurea, vista la forte dipendenza tra i due applicativi.

Il lavoro svolto si è dimostrato impegnativo in termini di tempo e impegno richiesto, in quanto si trattava di un progetto strutturalmente complesso con tanti requisiti da soddisfare. Uno di questi è stato la generalizzazione, poichè essendo inserito in un'architettura finalizzata al monitoraggio di strut-

ture diverse tra loro, l'applicativo doveva risultare idoneo ad ognuna di esse. Pertanto ho dovuto tener conto di tutte le possibili sfumature che possono presentarsi nei vari scenari presi in considerazione.
Fin da subito sono stato attratto dalla finalità di questa proposta progettuale: la prognostica, orientata a prevenire interventi di manutezione e al monitoraggio dello stato di salute di strutture e infrastrutture, è una tematica estremamente attuale che attraverso le tecnologie di cui disponiamo al giorno d'oggi possiamo affrontare con ottime probabilità di successo.
Per quanto riguarda gli sviluppi futuri del progetto, l'applicazione dovrà essere integrata con la parte progettuale relativa all'analisi sui dati, affidata al Politecnico di Milano. Ulteriori sviluppi futuri potrebbe riguardare l'estensione delle funzionalità di filtraggio dati, consentendo all'utente di integrare manualmente nell'applicativo modelli matematici da applicare sui dati; inoltre potranno essere aggiunte nuove tipologie di grafici al fine di visualizzare statistiche sui dati. Una funzionalità aggiuntiva che potrà essere sviluppata nel corso del tempo riguarda la possibilità di aggiungere e configurare dinamicamente nuove strutture da monitorare, definendo la composizione della rete di sensori e delle Thing.

Il lavoro non deve essere considerato definitivamente concluso, poichè sarà necessario mantenerlo costantemente allineato sia con l'evoluzione dello standard W3C che con gli scenari applicativi di monitoraggio che potranno presentarsi, in particolare in ambito industriale e civile.

Bibliografia

Articoli

[9] Francesco Lamonaca et al. "Internet of Things for Structural Health Monitoring". In: (apr. 2018), pp. 95–100. DOI: 10.1109/METROI4.2018.8439038. URL: https://www.researchgate.net/publication/330202313_IoT_for_Structural_Health_Monitoring.

[10] Shancang Li, Li Da Xu e Shanshan Zhao. "The internet of things: a survey". In: *Information Systems Frontiers* 17.2 (2015), pp. 243–259. URL: https://doi.org/10.1007/s10796-014-9492-7.

[14] D. Raggett. "The Web of Things: Challenges and Opportunities". In: *Computer* 48.5 (mag. 2015), pp. 26–32. ISSN: 1558-0814. DOI: 10.1109/MC.2015.149. URL: https://ieeexplore.ieee.org/abstract/document/7111885.

Online

[1] *About W3C*. URL: https://www.w3.org/Consortium/.

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

[3] *Forecast end-user spending on IoT solutions worldwide from 2017 to 2025*. URL: https://www.statista.com/statistics/976313/global-iot-market-size/.

[4] *Grafana Official Website*. URL: https://grafana.com/.

[5] *Internet of Things (IoT) connected devices.* URL: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/.

[6] *Internet of Things Global Standards Initiative.* URL: https://www.itu.int/en/ITU-T/gsi/iot/pages/default.aspx.

[7] *Kibana Official Website.* URL: https://www.elastic.co/kibana.

[8] Michael Koster. *W3C. Web of Things (WoT) Protocol Binding Templates.* 2020. URL: https://www.w3.org/TR/2020/NOTE-wot-binding-templates-20200130/.

[11] Gregg Kellogg; Pierre-Antoine Champin; Dave Longley. *W3C JSON-LD 1.1.* 2019. URL: https://www.w3.org/TR/2019/CR-json-ld11-20191212/.

[12] Zoltan Kis; Daniel Peintner; Johannes Hund; Kazuaki Nimura. *W3C. Web of Things (WoT) Scripting API.* 2019. URL: https://www.w3.org/TR/wot-scripting-api/.

[13] *Progetto Mac4Pro.* URL: https://site.unibo.it/mac4pro/it/descrizione.

[16] *W3C Working group.* URL: https://www.w3.org/WoT/WG/.

[17] *W3C. Participants in the Web of Things Working Group.* URL: https://www.w3.org/2000/09/dbwg/details?group=95969&order=org&public=1.

[18] *W3C. Web of Things (WoT) Architecture.* URL: https://w3c.github.io/wot-architecture/.

[19] *W3C. Web of Things (WoT) Servient Implementation.* URL: https://www.w3.org/TR/wot-architecture/#sec-servient-implementation.

[20] *W3C. Web of Things (WoT) Thing Description.* URL: https://www.w3.org/TR/wot-thing-description/.

Libri

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