Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Informatica

Relativistic Twin: Generazione Automatica di Digital Twin in un ambiente Web of Things

Relatore:
Chiar.mo Prof.
Marco Di Felice

Correlatore:
Dott. Luca Sciullo

Presentata da:
Mario Sessa

Sessione II - Secondo Appello
Anno Accademico 2021-2022

Alla mia famiglia
Ai miei amici
A tutti coloro che mi sostengono

Abstract

Con l'avvento dell'Industry 4.0, l'utilizzo dei dispositivi Internet of Things (IoT) è in continuo aumento. Le aziende stanno spingendo sempre più verso l'innovazione, andando ad introdurre nuovi metodi in grado di rinnovare sistemi IoT esistenti e crearne di nuovi, con prestazioni all'avanguardia. Un esempio di tecniche innovative emergenti è l'utilizzo dei Digital Twins (DT). Essi sono delle entità logiche in grado di simulare il reale comportamento di un dispositivo IoT fisico; possono essere utilizzati in vari scenari: monitoraggio di dati, rilevazione di anomalie, analisi What-If oppure per l'analisi predittiva. L'integrazione di tali tecnologie con nuovi paradigmi innovativi è in rapido sviluppo, uno tra questi è rappresentato dal Web of Things (WoT). Il Web of Thing é un termine utilizzato per descrivere un paradigma che permette ad oggetti del mondo reale di essere gestiti attraverso interfacce sul World Wide Web, rendendo accessibile la comunicazione tra piú dispositivi con caratteristiche hardware e software differenti. Nonostante sia una tecnologia ancora in fase di sviluppo, il Web of Thing sta giá iniziando ad essere utilizzato in molte aziende odierne. L'elaborato avrá come obiettivo quello di poter definire un framework capace di integrare un meccanismo di generazione automatica di Digital Twin su un contesto Web of Things. Combinando tali tecnologie, si potrebbero sfruttare i vantaggi dell'interoperabilitá del Web of Thing per poter generare un Digital Twin, indipendentemente dalle caratteristiche hardware e software degli oggetti da replicare.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

Oggigiorno, la continua digitalizzazione dei sistemi industriali e civili é diventato uno dei temi centrali di molte discussioni all'interno della nostra societá. In questi processi, non vi é dubbio che tecnologie come l'Internet of Things (IoT) hanno reso le procedure piú semplici; ma ha anche portato ad un aumento della complessitá dei sistemi man mano che il numero di dispositivi utilizzati era in aumento. Secondo studi recenti, il problema maggiore é dato dall'incompatibilitá tra differenti protocolli IoT con gli standard piú utilizzati; questo vincolo ha creato un freno alla rapida espansione dell'utilizzo dell'Internet of Things all'interno dei sistemi odierni. La difficoltá non si lega alla connettivitá tra dispositivi ad internet che, con l'assegnazione le procedure definite in TCP/IP o UDP/IP, ha portato ad un meccanismo semplice ed efficace per lo scambio di dati. Purtroppo, lo scambio di informazioni non garantisce che entrambi i dispositivi sappiano interpretare correttamente i dati ricevuti. Per questo motivo, i dispositivi IoT hanno avuto bisogno di un metodo di comunicazione basato su un protocollo universale, come HTTP. Tale protocollo é in grado di poter trasferire dati testuali, immagini, suoni o altri tipi di informazioni in maniera robusta ed efficiente; cosi che, varie Things possano elaborare ed interpretare i dati nella giusta maniera. Da tali principi, nasce il Web of Things (WoT). Il Web of Things é un insieme di standard definiti dal World Wide Web Consortium (W3C) per facilitare l'interoperabilitá, la frammentazione e l'usabilitá dell'Internet of Things. In altri termini, possiamo definire il Web of Things come un adattamento delle tecnologie IoT in fase di produzione, su standard come REST, HTTP e sistemi basati su URI, capaci di rendere possibile l'interazione tra differenti dispositivi con protocolli, gestione dati e hardware differenti. Parleremo am-

piamento del Web of Things nel capitolo due, cosi da fornire un'introduzione consistente alla base della proposta descritta nell'elaborato. Se il Web of Things risolve il problema dell'interoperabilitá, rimangono ancora vari difficoltá legate alla complessitá dei sistemi IoT moderni. L'eccessivo numero di dispositivi interconnessi puó portare problemi di gestione dati che possono creare, ad esempio: errori di assemblaggi nel settore del manufacturing, di lettura dati nei sistemi di monitoring o addirittura falsi allarmi in sistemi di emergenza che, nel giusto contesto, possono avere conseguenze vitali. Per poter risolvere questo tipo di problema, le aziende che forniscono servizi IoT provvedono a definire meccanismi di diagnostica capaci di rilevare possibili malfunzionamenti logici o fisici in modo di poter intervenire tempestivamente. Inoltre, é comune utilizzare sistemi di analisi, capaci di poter ottimizzare i processi di un sistema. Tali procedure si sono basate su simulazioni che, nei tempi piú recenti, si sono evoluti in Digital Twin. Sia le simulazioni che i Digital Twin rappresentano una copia logica di un sensore o di un dispositivo IoT. La differenza sostanziale si verifica sull'utilizzo dei dati reali. Mentre le simulazioni elaborano procedure basandosi su rilevamenti fittizzi, i Digital Twins creano un vero e proprio ambiente virtuale con un canale dati multidirezionale, in grado di fornire dati simulati, basandosi su un aggiornamento continuo adattatato su dati reali. In altri termini, i Digital Twin sono una rappresentazione logica di un dispositivo che non copia solamente le funzionalitá della Thing fisica, ma anche il suo comportamento. Combinando la potenza di calcolo aggiuntiva su un ambiente virtuale, i Digital Twin sono in grado di studiare piú problemi da punti di vista molto piú vantaggiosi rispetto alle simulazioni standard, con un maggiore potenziale di miglioramento per prodotti o processi. Integrando un sistema IoT con le tecnologie WoT e supportando i processi con applicazioni logiche basate su Digital Twis, i sistemi moderni possono evolversi in infrastrutture interoperabili, sicure ed efficienti. Tale soluzione é giá stata adottata da numerose aziende odierne per velocizzare il processo di innovazione in molti campi applicativi dell'IoT. Purtroppo, una delle maggiori difficoltá é data dalle multiple e costose procedure di generazione di Digital Twin, che orientano l'implementazione in base alle caratteristiche del dispositivo fisico, dei protocolli e dei dati che essi gestiscono. Questo approccio generativo ha come effetto collaterale lo sviluppo di modelli di Digital Twin fortemente orientati alle applicazioni che si interfacciano verso il singolo dispositivo. L'elaborato avrá come obiettivo quello di illustrare le principali tecnologie citate, in modo da garantire una co-

noscenza completa sull'ambiente sperimentato e, successivamente, si passerá alla descrizione del procedimento di generazione in modo da esporre l'intero processo generativo di una Digital Twins, astraendone le singole configurazioni tramite l'applicazione degli standard WoT, mostrando la progettazione, la validazione e un analisi sugli obiettivi ottenuti.

Capitolo 2

Stato dell'Arte

2.1 Internet of Things

Negli sviluppi recenti degli ultimi anni in campo di comunicazione e con il continuo evolversi delle tecnologie delle reti wireless, le applicazioni basate sull'utilizzo della rete hanno avuto una notevole evoluzione. Nello specifico, tecnologie come quelle basate sull'Internet of Things (IoT) hanno fornito un notevole incremento dei servizi in molti ambiti economici, industriali e sociali. I recenti sviluppi in campo di ricerca hanno portato alla realizzazione di infrastrutture come le Smart City [1], digitalizzazione del campo sanitario [2] ed ha rivoluzionato il campo industriale con la Industry 4.0 [3]. In questa sezione andremo ad analizzare il concetto di Internet of Things, fornendo una panoramica sulle caratteristiche che lo contraddistinguono dalle altre tecnologie e concluderemo con un analisi delle sue applicazioni piú recenti.

2.1.1 Concetto di Internet of Things

Il termine Internet of Things (IoT) non é recente, esso é stato definito da Kevin Ashton, cofondatore dell'Auto-ID Center del MIT, nel 1999 [4]; in stretta relazione con i dispositivi RFId (Radio Frequency Identification). Il concetto dell'IoT é un neologismo in stretta correlazione con il mondo delle telecomunicazioni. Nello specifico, con Internet of Things si fa riferimento ad un estensione delle tecnologie Internet verso gli oggetti e i luoghi dove essi sono localizzati. Con il termine "oggetto" si fa riferimento ad un dispositivo fisico (come sensori, attuatori, apparecchiature o sistemi complessi) in grado di interagire con l'ambiente circostante, connettersi con altri dispositivi e

compiere azioni in stretta correlazione con le caratteristiche dello spazio fisico dell'oggetto. Generalmente, quando si parla di sistemi IoT, si fa sempre riferimento ad una rete di oggetti intelligenti. Con oggetto "intelligente", si intende qualsiasi dispositivo elettronico o analogico equipaggiato con sensori e/o attuatori in grado di fornire una elaborazione dati per la comunicazione di informazioni o l'esecuzione di azioni in relazione all'ambiente. Ai sensori, va accoppiato un microcontrollore di dimensioni ridotte in grado di elaborare i dati dei sensori, per poter poi inviarli tramite i vari protocolli utilizzati per la trasmissione con altri dispositivi intelligenti all'interno di una rete IoT. Per poter svolgere tali attivitá, i dispositivi hanno generalmente un alimentazione, come batterie o prese elettriche. Una delle maggiori criticitá é la riduzione del consumo energetico che puó influeire sul ciclo di vita di un dispositivo, se gestita erroneamente. Per questo motivo, si costruiscono sistemi in grado di poter ottimizzare il consumo delle risorse, mantendo il sistema efficiente. [5]. Nel corso degli anni, l'Internet of Things ha avuto una crescita esponenziale nell'arco di 20 anni. Si prevede che il numero di dispositivi IoT aumenterá del 18%, fino a raggiungere 14.4 miliardi di dispositivi attivi per l'inizio del 2023 e una crescita del 22% negli anni successivi. Tali stime, peró, sono molto influenzate da eventi, come la crisi dei chip; che ha portato ad una notevole riduzione della produzione dei dispositivi elettronici. [6]

2.1.2 Caratteristiche

Le caratteristiche di un sistema IoT hanno fornito, negli anni, scelte progettuali basate sull'integrazione di servizi aggiuntivi per sistemi complessi e, da ció, anche un ottimizzazione per le procedure correlate. Tali caratteristiche possono essere sintetizzati nei seguenti punti [7]:

1. **Interconnettivitá:** I dispositivi IoT possono essere integrati in sistemi di comunicazioni moderne attraverso il supporto verso un ampio dominio di protocolli di comunicazione.

2. **Servizi legati alle Things:** Le Things possono essere gestite in relazione all'ambiente in cui sono state collocate, definendo informazioni e metadati necessari per una corretta fruizione dei servizi. Tali dati sono relativi non solo alle informazioni di ambiente, ma anche delle risorse, dello stato dei sensori e della gestione dei processi di un dispositivo.

3. **Eterogeneitá:** I dispositivi IoT possono comunicare tra loro anche se hanno caratteristiche hardware e software differenti, tramite meccanismi di proxying o adapter, per trasformare i dati di comunicazione in formati adeguati e comprensibili al loro ricevente.

4. **Dinamicitá di stato:** Un dispositivo IoT é capace di poter variare il suo stato in dipendenza di alcuni fattori, in maniera volontaria o involontaria. Esempi di variazione di stato possono essere l'accensione o lo spegnimento di un apparecchio elettrico, la connessione o la disconnessione alla rete internet, la gestione della frequenza di campionamento dei sensori e la variazione nei protocolli di comunicazione utilizzati.

5. **Scalabilitá:** Un sistema IoT é, generalmente, in grado di estendere i servizi andando ad aggiungere sensori e microcontrollori nell'ambiente di applicazione, gestendo i dati in maniera logica e in relazione alle funzionalitá del sistema.

6. **Sicurezza:** I dispositivi IoT possono scambiarsi o mantenere informazioni personali, é di buon uso adottare un approccio incentrato sulla sicurezza e la privatizzazione dei dati, in modo da non esporli ad agenti malevoli esterni.

7. **Connettivitá:** Il contesto IoT permette di abilitare l'accessibilitá ad una rete e la compatibilitá per la gestione dati da parte di un qualsiasi nuovo dispositivo autorizzato e connesso.

2.1.3 Architettura

Come mostrato in figura 2.1, l'architettura di un sistema IoT é divisa in strati logicamente correlati in modo da fornire un architettura modulare, scalabile e configurabile per differenti scenari [7]. Le caratteristiche degli strati possono essere elencate nei seguenti punti, seguendo un approccio bottom-up:

1. **Perception Layer:** Indica il livello fisico del dispositivo, é composto dai sensori e dai componenti fisici di un oggetto. La sua funzione é quella di gestire la lettura e l'immagazzinamento dei dati dei sensori. I tipi di dati variano a secondo delle funzionalitá del dispositivo (valori di temperatura, umiditá, posizione), per poi inviarli al network layer, in modo da poterli trasmettere verso gli altri dispositivi del sistema.

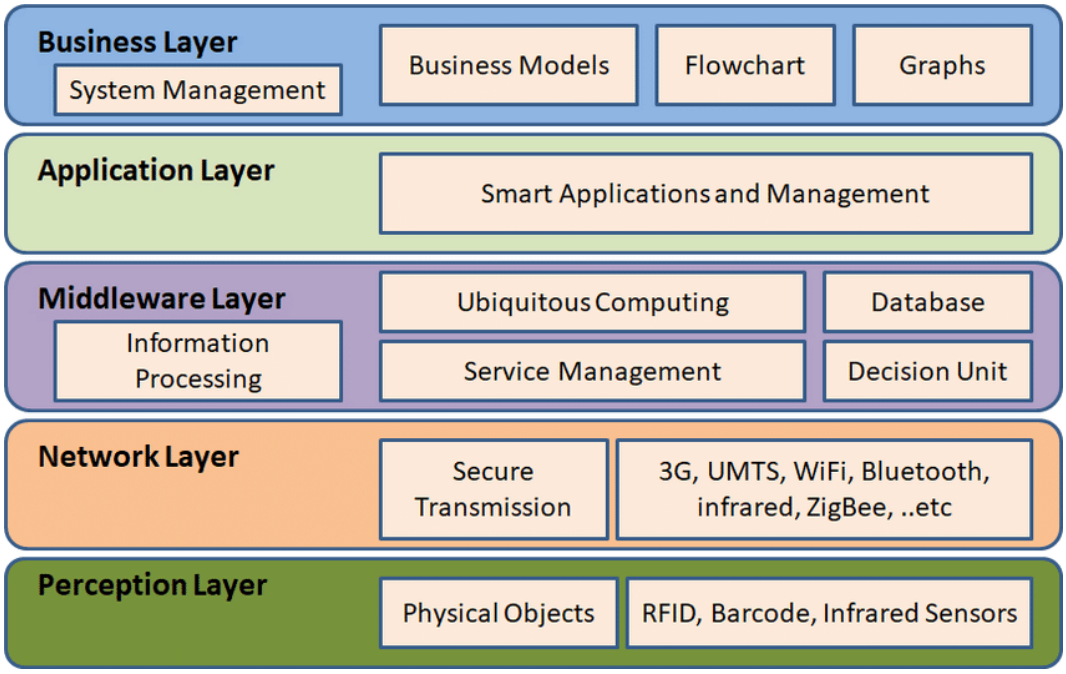

Figura 2.1: Architettura di un sistema IoT

2. **Network Layer:** Raffigura dispositivi, come i gateway, in grado di poter gestire il flusso di dati all'interno di una rete IoT. Tramite il network layer é possibile stabilire infrastrutture basate su comunicazioni per reti Machine-To-Machine (M2M), applicazioni context-aware o servizi IT. La comunicazione tra vari dispositivi é in relazione a connessioni in una rete locale (LAN) utilizzando segnali WiFi o Ethernet, oppure in una Personal Area Network (PAN) come per ZigBee, tecnologie Bluetooth Low Energy (BLE) e Ultra Wideband (UWB). Da ció, si ha la possibilitá di connettersi non solo con altri sensori, per definire dipendenze o influenzarne la logica di gestione dati, ma anche con dispositivi backend accedendo a Wide Area Network (WAN) come GSM, GPR o LTE.

3. **Middleware Layer:** Raffigura lo strato di gestione delle informazioni ottenute in processi di analisi e raccolta dei dati. Questo layer é uno dei piú importanti in termini di sicurezza; poiché puó contenere informazioni sensibili provenienti dalla trasmissione dati degli strati sottostanti.

4. **Application Layer:** Dalla raccolta dati dello strato middleware sottostante, é possibile stabilire un comportamento per rendere un dispositivo "intelligente". Questo comporta la gestione dati per fornire servizi in domini come le Smart City, Smart Building, Supply Chain, sistemi di allarme, monitoraggio o analisi.

5. **Business Layer:** Raffigura un layer facoltativo in grado di generare modelli come grafi e flowchart con lo scopo di rilevare problemi all'interno della comunicazione sulla rete IoT e provvedere un processo di ottimizzazione per aumentare l'efficienza complessiva dei processi logici del sistema. Tale livello é di particolare importanza per la gestione di processi decisionali incentrati sull'analisi dei Big Data e sulle loro criticitá [7, 8].

2.1.4 Campi di Applicazione

I campi di applicazione dell'IoT raffigurano un dominio in continua crescita. Questo aspetto é dovuto alla forte digitalizzazione di processi analogici della societá che possono portare ad un miglioramento delle performance in vari processi produttivi, analitici e predittivi [7, 9]. Alcuni dei settori che hanno tratto piú benificio nell'utilizzo o nell'integrazione con sistemi IoT sono:

1. **Smart City:** Raccogliendo dati da sensori collocati all'interno o all'esterno di un ambiente urbano, si possono definire dei modelli di business capaci di fornire servizi come la sicurezza urbana, la gestione automatizzata dei trasporti, il monitoraggio dei consumi energetici o altri processi che hanno come obiettivo quello di fornire un meccanismo in grado di migliorare lo stile di vita di un cittadino. [10]

2. **Automotive:** L'implementazione di sistemi e applicazioni integrabili con qualsiasi dispositivo elettronico, ha portato alla diffusione di processi IoT anche in relazione al monitoraggio delle performance dei veicoli. Questo puó avere notevoli vantaggi per rilevare guasti nel sistema del veicolo, in modo da ottimizzare la sicurezza del conducente e dei passeggeri, ma anche fornire un esperienza di guida ottimizzata, andando a fornire servizi di assistenza e di supporto in grado di rendere la guida di una automobile piú piacevole e leggera. Inoltre, molte Big

Tech si stanno sempre piú interessando alla digitalizzazione e alla fruizione di servizi IoT per le industrie automobilistiche, come, ad esempio: Google, Apple, BMW, Tesla o Mercedes. [12]

3. **Sanitá:** La diffusione di sensori e di attuatori, unita ad un affidabilitá di sistema considerevole, puó portare a molti benefici anche in ambito sanitario. Alcune applicazioni possono essere correlate allo stato di salute di un paziente in continuo monitoraggio, tramite sensori e dispositivi, come per i moderni peacemaker, o per la gestione di valori in terapie farmacologiche. [11]

4. **Industriale:** I processi produttivi industriali hanno avuto una delle piú grandi rivoluzioni tecnologie in relazione alla diffusione di applicazioni IoT; definendo le caratteristiche della Industry 4.0. Sono molti gli esempi di applicazione in questo ambito. Alcuni esempi possono interessare il rilevamento di errori di produzione in una catena di assemblaggio, monitoraggio di criticitá di un sistema o procedure di ottimizzazione. [13]

2.2 Web of Things

Nei sistemi IoT moderni, gli sviluppatori devono interfacciarsi con molti ambienti complessi. Uno dei maggiori problemi, come detto nel capitolo precedente, é strettamente legato alle tecnologie eterogenee negli ambienti IoT esistenti. Essi offrono servizi con tecnologie e protocolli di comunicazioni eterogenei e, nella maggior parte dei casi, non compatibili e poco integrabili. Questa diversitá include variazioni nelle configurazioni hardware e software, nei modelli dei dati, nella formattazione dei payload nei messaggi e perfino nella sicurezza. Le applcazioni IoT sono spesso sviluppate andando ad usare tecniche di implementazioni fortemente dedicate ad un caso d'uso specifico. Questo aspetto ha portato allo sviluppo di applicazioni IoT che hanno lo svantaggio di non essere riadattate per un uso generico, rendendo le Thing orientate ad un uso specifico e poco interoperabile [14]. Durante il ciclo di vita, queste applicazioni hanno varie problematiche di estendibilitá, manutenibilitá e di riuso. A tale scopo, il Web of Things é una soluzione di base definita come un set di standard che integra i vantaggi del Web a sistemi IoT in fase di sviluppo o in produzione. Alla base dell'integrazione, vi sono

dei building block che aiutano a semplificare le applicazioni IoT seguendo le regole dei paradigmi web moderni. Questo approccio ha come obiettivo la trasformazione dei servizi IoT in sistemi flessibili ed interoperabili; andando a poter effettuare procedimenti di integrazione in grado di far comunicare applicazioni di domini differenti tramite un meccanismo standarizzato. In altri termini, il WoT puó essere visto come un ottimo strumento commerciale per rendere i sistemi IoT, quando se ne necessita, piú simili a vere e proprie applicazioni Web. Nelle sezioni successive, andremmo a determinare l'architettura di base del Web of Things e come poter sfruttare i vari meccanismi del paradigma per poter trasformare i sistemi IoT moderni. Questo procedimento sará alla base della proposta di elaborato poiché, con l'introduzione di sistemi basati su Digital Twins, si potranno creare automaticamente delle applicazioni in grado di gestire o analizzare qualsiasi Things presente nel campo dell'Industria 4.0.

2.2.1 Architettura

Quando si parla di architettura del Web of Things, non si fa riferimento ad una implementazione vera e propria ma solamente alla descrizione dei meccanismi logici implementabili a partire da una Thing di un applicazione in stato di sviluppo o in produzione. Una Thing raffigura una astrazione che puó essere fisica (rappresentata dal singolo sensore, attuatore o gruppi di dispositivi che interagiscono con l'ambiente in cui sono immersi), oppure virtuale. Tale Things rappresentano una descrizione dei metadati del device a cui sono associati, fornendo le caratteristiche di comunicazione e le configurazioni hardware e software. Nel paradigma Web of Things fornito dalla W3C, la descrizione dei metadati di una Thing Web é formalizzata all'interno di un documento chiamato Thing Description (TD). Ogni entitá consumer, che interagisce con la Thing WoT, deve poter processare la Thing Description prima di interagire con il dispositivo vero e proprio; questo perché la TD ha il compito di dover fornire tutte le informazioni di interfacciamento e processamento dei dati di comunicazione. Se un consumer non ha tale requisito, non avrá noto quali dati fornisce la Thing e come adoperare i meccanismi di comunicazioni (protocolli, payload, ecc.). Il meccanismo di parsing di una Thing Description avviene generalmente tramite formati noti nel contesto del Web. In maniera ufficiale, si utilizzano file in fomato JSON che sono in grado di essere processati tramite i classici strumenti web oppure tramite processori

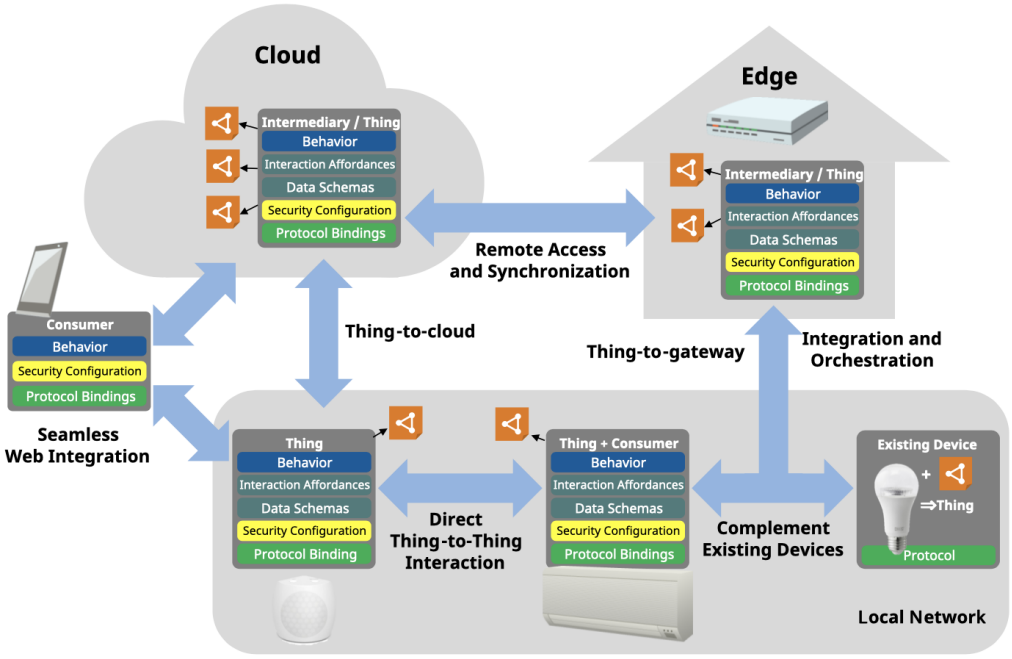

Figura 2.2: Architettura Astratta del W3C WoT con differenti interazioni tra differenti componenti di un sistema IoT-based.

JSON-LD[17] in grado di sottolineare dei modelli informativi, cosi da creare un modello orientato a grafo compatibile con la serializzazione dei documenti JSON-LD 1.1. Inoltre, tale procedimento risulta essere facilmente integrabile con tecniche di Web Semantic, l'applicazione di inferenze semantiche o accoppiamenti di processi in termini ontologici, in grado di rendere l'interazione dei consumer autonomi. Bisogna specificare che una Thing Description é alla base dell'architettura del Web of Things e fornisce uno strato di interfacciamento esterno fortemente dedicato alla Thing a cui fa riferimento. In altri termini, non possiamo definire una Thing Description come un blueprint di un tipo di Thing (o dei soli tipi di dati che essi processano) ma un insieme di informazioni fortemente orientate al singolo comportamento del dispositivo preso sotto analisi. Una Thing Description é una componente dell'architettura paragonabile ad una pagina HTML fornita da un web service. Ogni Thing, nel contesto WoT, deve essere rappresentata da almeno una Thing Description in modo da poter adoperare una procedura di riconoscimento in grado di poter fornire una rappresentazione machine-understandable; questo

permette ai Consumer di scoprire ed interpretare le capacitá e il comportamento di una Thing tramite annotazioni semantiche, in modo da poter adattarle a differenti implementazioni (ad esempio sfruttando differenti protocolli di comunicazione, mantenendo lo stesso flusso dati). Grazie alla Thing Description, si ha non solo la possibilitá di garantire una interoperabilitá tra differenti entitá di un sistema IoT, ma permette anche di poter fornire informazioni sul comportamento di una Thing é formalizzare un Digital Twins in maniera semplice e diretta. Nelle prossime sezioni andremo a trattare le principali componenti del Web of Things, descrivendolo in un quadro generale capace di poter far capire come si presenta l'ambiente di sviluppo su cui si baseranno alcuni moduli del progetto.[15]

2.2.2 Web Things

Una Web Things raffigura l'utilizzo astratto di un dispositivo andandone a specificare alcune caratteristiche fondamentali come: il suo comportamento, l'interaction affordance, le configurazioni di sicurezza, il protocol bindings per la mappatura verso configurazioni di protocolli, tipi di dati e messaggi specifici. In altri termini, possiamo definire una Web Things come un insieme di informazioni e metadati che vanno ad interfacciarsi con il modello di interazione di una piattaforma IoT. Per poter interagire, i vari consumer devono conoscere quali siano le caratteristiche della Things; cosi da adattare la comunicazione in modo da rendere l'ambiente interoperabile. Per farlo, si utilizza un documento che racchiude tutti i metadati: la Thing Description (TD). A tale concetto se ne aggiungono altri che andranno a definire le fondamenta delle Web Things. Tali componenti sono chiamati Building Blocks. Il loro utilizzo é fortemente orientato a stabilire delle caratteristiche di interoperabilitá tra differenti modelli di iterazioni; cosi da poter far interagire piú dispositivi mantendo un livello di gestione astratto e compatibile con le maggiori configurazioni utilizzate nei sistemi IoT odierni.

Interazioni nel Web Things

Come anticipato nella sezione precedente, il comportamento astratto di una Thing del sistema viene definito all'interno di un documento ben formattato chiamato Thing Description. Ad alto livello, la TD rappresenta una

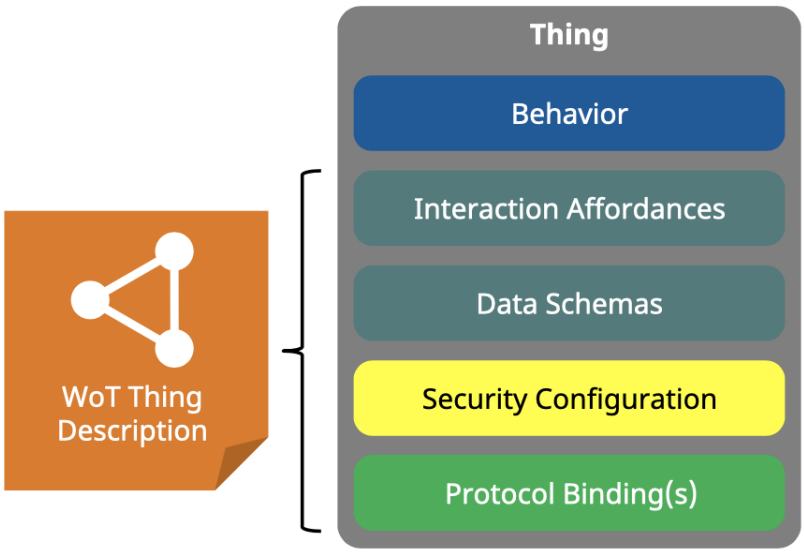

Figura 2.3: Composizione di una Web Things e delle informazioni contenute all'interno della Thing Description affiliata.

interfaccia in grado di far conoscere a nodi esterni quali siano le funzionalitá, i vincoli e le caratteristiche proprie di una Thing. Nello specifico, una Thing produce una Thing Description esponendola al sistema che ne sfrutta le informazioni per poter determinare come interagire, definendo, ad esempio: quale protocollo utilizzare, la formattazione dei dati e dei messaggi. Un nodo consumer avrá l'obiettivo di processare le Thing Description per poter poi adattare l'interazione, rendendo cosi possibile lo scambio di messaggi e di risorse. Tale sistema segue, logicamente, lo stesso processo utilizzato nelle pagine HTML fornite da server Web all'interno di un web browser di un utente. Infatti, i meccanismi di discovery delle Thing Description sono orientate a directory, proprio come per la navigazione nei file system o negli URL per un endpoint web. Secondo gli standard W3C WoT, una Thing é una rappresentazione astratta di una entitá virtuale all'interno di un sistema IoT. Questo significa che non solo puó essere associata ad un sensore o un attuatore, ma anche a gruppi di sensori che sono logicamente correlati. Per questo motivo, una Thing Description puó far riferimento alle caratteristiche di piú dispositivi fisici. Inoltre, per ridurre la complessitá di implementazione, é possibile definire delle Thing Description gerarchiche che, tramite l'utilizzo

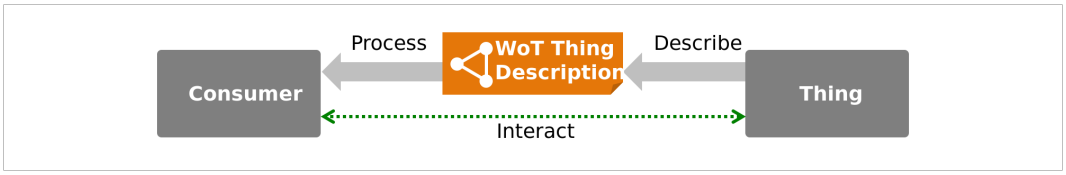

Figura 2.4: Interazione tra una Thing e un Consumer

di link, si possono referenziare piú Thing Description in modo da esporne un'astrazione che mantiene una struttura di dipendenze logiche interne tra piú sensori o risorse. Esempi di questo genere possono essere la definizione di Thing Description inerenti a sensori di temperatura e di umiditá all'interno di una stanza; essi possono far riferimento ad una singola Thing Description o multipli documenti esposti all'interno di referenze di una TD generica. In altri termini, possiamo trattare le Thing su piú livelli di astrazione, indipendentemente dalla complessitá, dal numero e dal tipo di dispositivi a cui si fa riferimento. Infine, una delle caratteristiche nella gestione delle reti sul Web é data da scenari in cui non vi é la possibilitá di poter raggiungere e interfacciarsi ad Internet dalla propria rete locale, generalmente questi problemi sono dati per vincoli di sicurezza dei firewall di dispositivi utilizzati oppure per problemi NAT (Network Address Translation). Nel web, possiamo trovare varie soluzioni che rendono interoperabili e accessibili le reti, un esempio é l'utilizzo di proxy server che hanno la funzione di intermediari di dati da e verso una rete locale. Tipicamente, in uno scenario di questo genere possiamo considerare i proxy server come un ponte dati che unisce piú reti, in grado di stabilire un canale bidirezionale e, in alcuni casi, aggiungere funzionalitá durante il processamento dei messaggi. Dato che possiamo definire qualsiasi risorsa come una Thing, anche i proxy possono essere definiti come tali. Il comportamento di un proxy server o di altri tipi di entitá intermediarie possono essere astratte attraverso un WoT Thing Description generico. La differenza tra una WoT TD di un intermediario rispetto a quella del singolo dispositivo é data dall'interfacciamento WoT che provvede alla referenziazione di piú Thing non direttamente raggiungibili da un consumer esterno. In altri termini, un consumer puó utilizzare una o piú Thing indirettamente andando ad adattare l'interazione diretta con un nodo intermediario. Dal punto di vista logico, non vi é nessuna distinzione tra l'interazione diretta e indiretta con una Thing, questo perché l'astrazione della WoT TD permette l'interfacciamento indiretto verso piú entitá simultaneamente; senza sapere

in maniera specifica se una Thing faccia riferimento ad un singolo, ad un gruppo di dispositivi oppure ad un loro intermediario. Dal punto di vista dell'intermediario, le Things che espone vengono consumate per poter riconoscere le configurazioni interne, questo genererá Thing e TD piú generiche all'interno del proprio WoT runtime in modo da rendere visibile l'interfacciamento delle Things esternamente alla rete. Tale architettura gerarchizzata permette di estendere un concetto astratto verso piú entitá su differenti layer logici di rete, andando ad integrare piú sistemi (System of Systems o SOS) e dispositivi eterogenei tra di loro.

2.2.3 Building Blocks

All'interno di questa sezione, andremo ad illustrare i principali concetti alla base dell'architettura Web of Things, specificando nel dettaglio anche alcune definizioni precedentemente utilizzate.

WoT Thing Description Una Thing Description rappresenta uno dei principali building block nel Web of Things. Essa contiene tutti i metadati e le caratteristiche configurabili di una Thing andando ad esporla in modo da poter garantire un accesso alla comunicazione, andando a conoscere, ancor prima dell'interazione, tutte le sue caratteristiche. Una Thing Description non é altro che un documento ben formattato secondo uno schema logico preciso. Esso puó essere esposto sia internamente alla Thing che rappresenta o esternamente (questo secondo caso é abbastanza utilizzato soprattutto per tipi di dispositivi che hanno limitazioni in termini di risorse e memoria). Secondo lo standard W3C WoT, le Thing Description sono formate da 4 componenti principali:

1. **Metadati:** raffigurano delle informazioni testuali descrittive in grado di associare alla Thing un significato e un contesto semantico di utilizzo, senza approfondire sulla configurazione hardware e software. Tale componente é una delle principali e ha varie informazioni, tra cui: titolo, ID, descrizione, dati di creazione, di ultima modifica e di versione.

2. **Interaction Affordance:** rappresentano un insieme di informazioni utili al consumer per poter gestire l'interazione con una WoT Thing. Sono formati da informazioni quali: titolo, descrizione e affordance

per le proprietá, azioni ed eventi. Parleremo nello specifico delle varie tipologie di affordance successivamente.

3. **Schema Dati:** Rappresenta tutte le informazioni relative alla struttura dei payload supportati dalla Thing, i dati con i rispettivi tipi che vengono inviati o ricevuti dal dispositivo associato alla Thing e le informazioni scambiati all'interno della comunicazione.

4. **Public Security Metadata:** sono dati relativi alla configurazione dei layer di sicurezza di una Thing andando a descriverne tutti i meccanismi utilizzati per proteggere la comunicazione e i dati forniti o ricevuti, i diritti di accesso che si necessitano per poter effettuare una comunicazione autorizzata, informazioni relativi a protocolli di sicurezza, come le chiavi pubbliche, in modo da permettere a chiunque conosca la TD di poter non solo comprendere e comunicare i dati alla Web Thing ma anche rispettarne i protocolli di sicurezza.

5. **Protocol Binding:** tali dati hanno lo scopo di concretizzare i concetti descritti all'interno delle informazioni della Thing Description. In altre parole, hanno il compito di poter associare i moduli relativi a protocolli concreti con i comportamenti e le interazioni (Affordance) previste dalla TD. Lo standard W3C WoT serializza il Protocol Binding come hypermedia controls, che hanno il compito di fornire meccanismi machine-understadable in grado di poter far comprendere come attivare un affordance. Parleremo nel concreto di come avviene il protocol binding nella sezione successiva.

In conclusione, la Thing Description é il principale building block secondo gli standard W3C WoT che ha principalmente due scopi: il primo é che i TD permettono la comunicazione machine-to-machine (M2M) tra piú Things, il secondo é che fornisce un documento comune e con un formato uniforme per gli sviluppatori. Inoltre, esso puó fornire tutti i dettagli necessari per creare un applicazione in grado di accedere ai device IoT raffigurati logicamente dalle Web Things.

All'interno dell'elaborato, la presenza delle WoT Thing Description risultano essere molto utili per poter determinare quale sia la struttura logica

Figura 2.5: Snippet di Thing Description per il controllo di un LED

iniziale che una Twin dovrebbe assumere. Infatti, essa fornisce dati inerenti alla configurazione e alla logica dietro i dati che processa un particolare dispositivo associato come, ad esempio, il protocollo utilizzato e i tipi di dati di input. A questo verranno aggiunte delle informazioni aggiuntive che si allontaneranno dallo standard W3C per definire un modello comportamentale della Web Thing; l'estensione dei metadati verrá largamente descritta nella sezione 3.4. Infine, l'accesso alle Thing Description avviene tramite differenti metodi che, come URL per le pagine web, sono facilmente raggiungibili e

utilizzabili da qualsiasi entitá del sistema e in qualsiasi layer dell'architettura (consumer esterni, cloud, edge gateway o dispositivi sulla rete locale). [16]

WoT Interaction Affordances Uno degli aspetti centrali del W3C WoT é quello di poter definire il comportamento di una Things tramite dei metadati in grado di specificare quali azioni poter adoperare per interagire con un dispositivo. Tali informazioni sono descrittive, ció permette di far si che un consumer riesca ad identificare cosa una Thing sia capace e come provvede ad effettuare le operazioni permesse: tali informazioni vengono chiamate Interaction Affordance. Il termine "affordance" ha una natura nella psicologica, ma é d stata adottata per gli studi di interazione uomo-macchina (HCI, Human-Computer Interaction), basandosi sulla definizione data da Donald Norman. Per D. Norman, l'affordance é riferita ad una proprietá percettiva di una oggetto in grado di essere manipolata per cambiarne lo stato. Nel Web of Things, le affordance fanno riferimento alle caratteristiche del modello di interazione, ossia specificano quali e come una Thing puó elaborare dati. Nello specifico, le Interaction affordance si dividono in tre differenti tipi:

1. **Proprietá:** fanno riferimento allo stato di una Thing. Lo stato esposto é una proprietá leggibile in grado di poter far si che un consumer esterno possa sempre tener presente quale sia il valore attuale di una Thing durante le interazioni. Allo stesso modo, vi sono alcune proprietá che possono essere modificate direttamente tramite operazioni di scrittura. Vi é la possibilitá di poter rendere alcune proprietá osservabili, ció sta a significare che, al cambio di stato, un consumer riceverá anche i dati relativi al cambio dei valori di una proprietá modificata della Thing. Esempi di proprietá possono essere i valori dei sensori legati ad un microcontroller (temperatura, pressione, umiditá, ecc.), lo stato d'ambiente di un attuatore oppure dati risultanti da una computazione.

2. **Azioni:** Se le proprietá raffigurano lo stato di una Thing, le azioni sono tutte quelle funzioni che, se invocate, possono manipolare direttamente o indirettamente lo stato esposto da uno o piú dispositivi. Una singola azione puó manipolare una o piú proprietá contemporaneamente; inoltre, é in grado di modificare non solo lo stato logico di una Thing (come i valori logici o metadati di un dispositivo) ma anche lo stato fisico (ad esempio la posizione di un attuatore nell'ambiente in cui é immerso).

3. **Eventi:** Gli eventi raffigurano delle particolari condizioni legate allo stato di una Thing indipendente dalle azioni adoperate. In altri termini, raffigura un procedimento asincrono in grado di essere gestito secondo procedure specifiche invocabili indirettamente e, in alcuni casi, sono anche in grado di passare informazioni o istruzioni ad uno o piú consumer con cui si sta interagendo. Esempi di eventi possono essere collegati a sistemi di allarmi invocabili quando si verifica una particolare condizione di stato oppure allo scadere di un timeout (eventi periodici).

Poiché nelle Interaction Affordance vi si ha la manipolazione dei dati, si ha bisogno di specificare anche lo schema a cui fanno riferimento. Questi possono essere definiti sia internamente alle specifiche delle Affordance, sia esternamente nei media type nel Protocol Binding descritto nella sezione successiva.

WoT Binding Templates

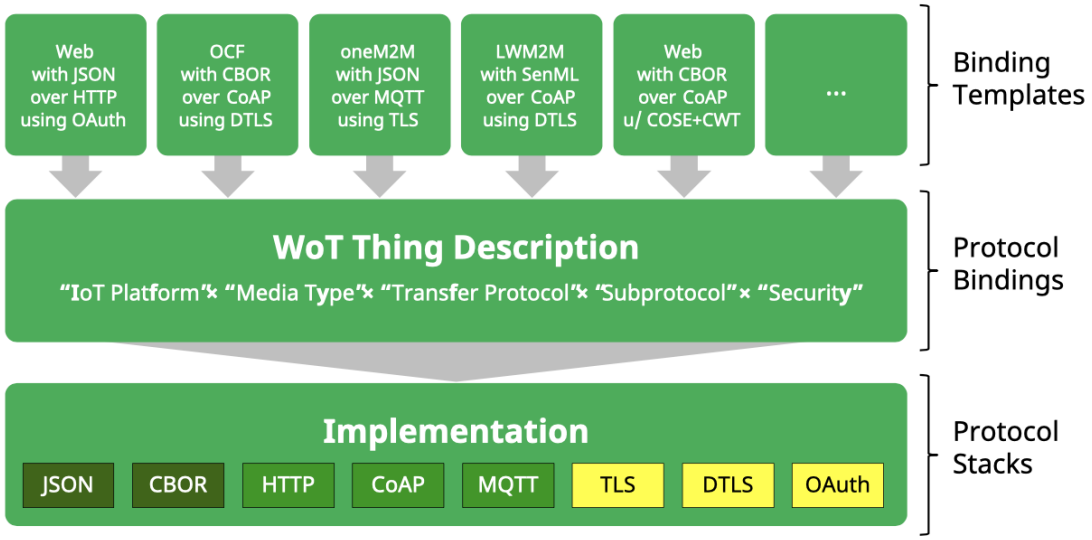

Figura 2.6: Procedimento di mappatura tramite il Protocol Bindings di Binding Templates con le rispettive implementazioni.

Le WoT Thing Description hanno come obiettivo finale quello di dover esporre un modello di interazione astratto su cui potremmo basarci per la generazione di Digital Twins e, allo stesso modo, poter capire il comportamento dei dispositivi associati ad una Thing. Questo permette di poter definire applicazioni in grado di scalare indipendentemente dalle singole configurazioni dei dispositivi. Con l'introduzione al Web of Things, la comunicazione tra piú Things viene vista in un contesto Web. Questo fa si che si possano sviluppare applicazioni in grado di supportare protocolli e standard a livello di rete, in grado di poter far interagire piú dispositivi allo stesso modo dei nodi su Internet. Attualmente, le piattaforme IoT hanno vari protocolli allo strato di rete e di trasporto per la gestione dell'invio e la ricezione di messaggi, l'obiettivo dei Binding Templates é quello di fornire un insieme di metadati e di blueprint capaci di poter definire le istruzioni di iterazioni per la comunicazione con le differenti piattaforme IoT (OCF9, oneM2M10, Mozilla IoT11, ecc.). La quantitá di Thing presenti in tali sistemi é enorme, raggiungendo perfino sistemi applicativi con piú di un milione di terminali e sensori. Inoltre, i vari standard stanno convergendo sempre piú verso un insieme comune di protocolli che, peró, possono avere configurazioni peculiari e con differenze in varie caratteristiche: formati, schema dei payload, tipi di dati, composizione dei messaggi. Generalmente, le Thing Description non riescono a risolvere questo problema direttamente; questo perché costituiscono solamente un interfaccia di esposizione delle Things, senza peró gestirne l'implementazione o problemi legati all'interoperabilitá. A fronte di questo ostacolo, nel contesto WoT si é creato meccanismo di Binding Templates. Tali processi hanno come obiettivo quello di inserire istruzioni sull'abilitazione dei protocolli descritti nelle WoT Thing Description; in grado di fornire una interfaccia verso un contesto web supportato. Essi permettono di mappare lo scheletro astratto delle configurazioni di un dispositivo con implementazioni concrete in grado di poter specificare i tipi di protocolli utilizzati (es. HTTP, MQTT, COAP), l'architettura sottostante (es. PubSub o RESTful) e della formattazione dei messaggi; seguendo sempre una logica di iterazione ben nota nelle interaction affordance del TD. Per poter adoperare il Protocol Bindings, si ha una mappatura tra le informazioni del TD e le infrastrutture implementate che rispettano i requisiti specificati dal dispositivo [18]. In altri termini, tramite i Binding Templates, un consumer puó comunicare ad alto livello con una Thing andando a richiamare le operazioni permesse dalle interaction affordance in contesti web (ad esempio utilizzando metodi

GET, POST, PUT, DELETE) e, a secondo del tipo di dispositivo che espone i servizi della Thing, andare ad adattare il modello di iterazione astratto con un implementazione che permette di comunicare in maniera corretta con qualsiasi tipo di dispositivo. Le configurazioni e i meccanismi connessi ai Binding Templates si legano ai metadati su 5 dimensioni:

- **Piattaforme IoT:** Si provvede a delle modifiche interne ai protocolli definite internamente al sistema in cui vengono utilizzate. Esempi possono essere variazioni di header HTTP, modifiche sulle configurazioni dei messaggi o delle opzioni in CoAP. Tali cambiamenti devono essere descritti all'interno di campi "form" del TD a cui si fa riferimento.

- **Media Type:** Specifiche che possono essere aggiunte durante il meccanismo di Binding Templates inerente all'introduzione di informazioni legate alle rappresentazioni e serializzazione dei dati durante lo scambio di informazioni. Specificano il formato e la gestione dei dati durante la comunicazione.

- **Transfer Protocol:** Specifiche sul protocollo applicativo utilizzato (MQTT, CoAP, AMQTP, HTTP).

- **Sub-Protocol:** Opzioni o specifiche aggiuntive ai Transfer Protocol utili per poter interagire. Esempio di sub-protocol sono configurazioni di long-polling, web sockets o altre configurazioni che modificano alcune proprietá di comunicazione a partire da un protocollo standard.

- **Security:** Configurazioni aggiuntive per garantire l'affidabilitá e protezione della comunicazione dati in vari livelli dello stack dell'architettura di rete implementata.

WoT Scripting API Nel contesto Web of Things vi sono anche building blocks opzionali. Essi garantiscono ulteriori servizi che andranno ad ottimizzare la gestione dello scambio di messaggio e di dati nel modello di interazione astratto. Nella documentazione ufficiale W3C WoT, si definisce col termine WoT Scripting API: una interfaccia di programmazione che permette di trovare e consumare Thing Description in maniera programmatica tramite l'utilizzo di script. Tali script vengono eseguiti all'interno di un WoT Runtime in grado di definire oggetti locali collegate con il comportamento di una

Thing remota. All'interno dell'instanza della Thing locale, vengono specificate tutte le proprietá logiche della Thing remota, tra cui anche le interaction affordances (proprietá, azioni ed eventi). Inoltre, si da la possibilitá di instanziare sia delle Consumed Things e gestirne le interazioni o definire delle Exposed Things che forniscono servizi ad entitá esterne. In altri termini, tramite le WoT Scripting API si ha un interfacciamento a partire da script eseguiti in un WoT runtime in grado di esporre e consumare Things secondo un livello di astrazione locale. Generalmente, le scripting API vengono implementate in nodi di rete in cui non vi sono vincoli relativi alle risorse utilizzate, tipi di dispositivi di questo genere sono, ad esempio, i gateway di rete. Possiamo definire le WoT Scripting API come delle API ECMAScript[1] di basso livello che eseguono concretamente le direttive date dalle caratteristiche di una Thing Description. Gli script relativi a tali implementazioni non sono vincolanti ad un singolo linguaggio. All'interno della WoT Runtime possono essere traducibili script appartenenti a linguaggi differenti da quelli ECMAScript come: Java, Python o C++. Questa caratteristica permette di eseguire un ambiente runtime interoperabile andandone ad evitare limitazioni inerenti al linguaggio di programmazione utilizzato. L'unico vincolo sull'implementazione di tali script é relativo ai processi logici forniti all'interno delle direttive della Thing Description. In concreto, ogni nodo puó sfruttare all'interno del proprio WoT Runtime delle implementazioni di Thing esposte o consumate a partire dalle caratteristiche fornite nelle TD. Ogni nodo, quindi, puó consumare ed esporre piú di una Thing nel proprio ambiente locale. [19] Nello specifico, possiamo riassumere le Scripting API attraverso l'implementazione di 3 tipi di componenti principali:

1. **API WoT:** Un oggetto che viene esposto seguendo uno dei design pattern piú utilizzati nel contesto web: Il Singleton. Ha come obiettivo principale quello di scoprire delle TD per poter generare Thing da esporre o consumare.

2. **ConsumedThing:** A differenza dell'oggetto API WoT, la ConsumedThing é un interfaccia che viene utilizzata per adattare classi Thing seguendone uno schema rigido in grado di definire operazioni quali: lettura e scrittura di proprietá, invocazione di azioni, sottoscrizione o cancellazione ad eventi.

[1] https://tc39.es/ecma262/

3. **ExposedThing:** Interfaccia in grado di specificare degli oggetti Thing con vari handler per poter gestire richieste inerenti a processi o richieste di scrittura e lettura di proprietá, invocazione di azioni o gestione di eventi. A differenza del ConsumedThing, ha la funzione di servire servizi ad entitá Consumer esterne.

WoT Security Uno degli aspetti piú critici nei sistemi IoT é la sicurezza, questo ha portato alla creazione di linee guida sulla sicurezza abbastanza rigide che, se non rispettate, rendono la comunicazione funzionalmente impossibile. La sicurezza nel WoT é un problema trasversale presente in ogni strato dell'architettura, essa diventa di vitale importanza in casi d'uso in cui vi é un trattamento di dati sensibili o privati. A fronte di questo problema, gli standard definiti dalla W3C specificano delle linee guida da dover seguire durante l'implementazione di sistemi basati sull'infrastruttura Web of Things. Il documento e le linee guida di sicurezza hanno solamente uno scopo informativo, al fine dell'elaborato non ci soffermeremo sulla loro gestione. Peró, per scopi informativi, é di buon uso introdurre le caratteristiche principali in modo da avere un quadro generale dell'infrastruttura anche sotto un punto di vista di sicurezza. Nello specifico possiamo sintetizzare i concetti delle linee guida nei seguenti punti:

1. Sicurezza sulle TD: Fanno riferimento a tutte le pratiche di protezione dei dati di una Thing Description e della Thing collegata. Specificano delle pratiche di sicurezza nella progettazione, utilizzo e diffusione delle Thing Description visibile alle sole entitá principali, limitandone gli accessi e vincolando le condivisioni. Inoltre, vi sono riferimenti riguardo la protezione dei dati sensibili, privati e immutabili all'interno di una TD.

2. Sicurezza sui Dati sensibili: Utilizzo di protocolli con comunicazioni sicure come, ad esempio: TLS con HTTP (HTTPS), CoAP con adattabilitá DTLS (CoAPS) e MQTT con TLS (MQTTS). Inoltre raffigurano anche linee guida per la definizione di un documento informativo per gli utenti finali sul trattamento dei dati, specificando quali informazioni private vengono esposte durante la comunicazione.

3. Sicurezza nella progettazione delle interfacce WoT: Si basa l'implementazione su paradigmi AAA (Authentication, Authorization e Accoun-

ting) definendone gli schemi. Minimizzano le funzionalitá e la complessitá delle interfacce in modo da renderle maggiormente gestibili. Infine, definiscono istruzioni per eseguire, ad ogni patch, sessioni di validazione e di testing in maniera corretta e completa.

Le specifiche di sicurezza, con annessi protocolli e meccanismi AAA, vengono specificati all'interno della Thing Description. [20]

2.2.4 WoT Servient

Nella sezione precedente abbiamo definito i vari buiding block messi a disposizione dall'astrazione fornita dallo standard W3C WoT. In questa sezione andremo ad analizzare una componente che, partendo dai vari building block, ha il compito di gestire e definire l'ambiente di esecuzione del sistema: il WoT Servient. Un Servient é rappresentato da uno stack software con il compito di implementare le astrazioni dei building block. Nello specifico,

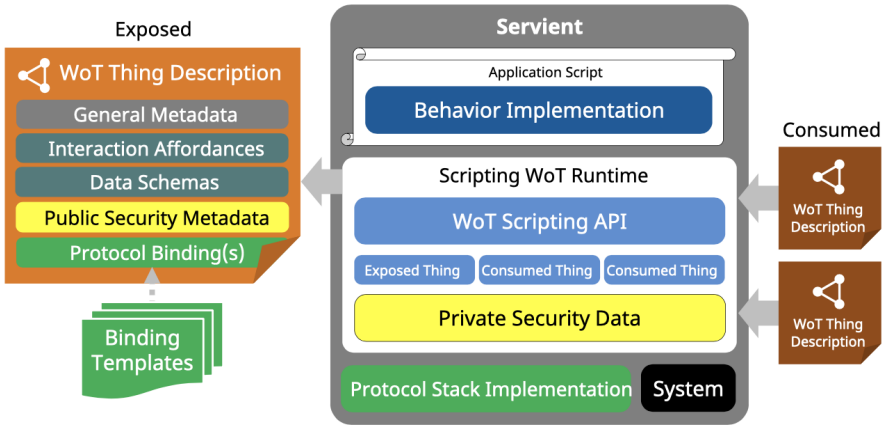

Figura 2.7: Stack di un WoT Servient che utilizza le WoT Scripting API

esso puó caricare una Thing Description, esporre e consumare Web Things, andando ad implementare le loro interfacce, seguendo le regole definite nei Binding Templates di una specifica Thing Description. Un Servient si puó considerare una entitá costruttice per componenti e ambienti di una rete Web of Things.

Lo stack software che lo compone é definito dai seguenti moduli:

1. **Application Script:** Raffigura il piú alto livello dello stack di un Servient, esso é composto dall'implementazione del comportamento logico di una Thing, andando a definire le funzioni che gestiscono e implementano la logica delle Interaction Affordance. Nello specifico, in dipendenza dal tipo di Thing di riferimento, l'application script definisce il comportamento di un Consumer, di una Thing o di un intermediario (Proxy). L'implementazione si puó appoggiare su un implementazione nativa delle API WoT o sulle WoT Scripting API.

2. **Scripting WoT Runtime:** Raffigura l'ambiente runtime in cui vengono mantenute le Thing esposte o consumate, esse vengono implementate secondo le interfacce definite nella Scripting API o, in alternativa, ad API native che rispettino l'astrazione WoT. All'interno del WoT runtime si ha la gestione del ciclo di vita di una Thing.

3. **Protocol Stacks:** Raffigura moduli che definiscono l'implementazione delle funzionalitá di base per la comunicazione in differenti protocolli come, ad esempio, HTTP, HTTPs, CoAP, MQTT o simili. Possono fornire anche differenti implementazioni per un medesimo protocollo, differenziando caratteristiche come il parsing, la serializzazione o la gestione dei messaggi in vari formati (JSON, CBOR, XML) andando a differenziare i vari media type utilizzabili.

4. **System API:** Definisce un interfaccia in grado di far comunicare i moduli dello stack con l'hardware di sistema per accedere a risorse come la memoria. Anche se fa parte dello stack di un WoT Servient, le sue funzionalitá sono gestite da API esterne allo standard fornito da W3C WoT.

2.3 Digital Twins

In questo capitolo, andremo ad analizzare la letteratura dedicata al concetto di Digital Twins per definirne il concetto, il contesto di applicazione e i vantaggi d'uso.

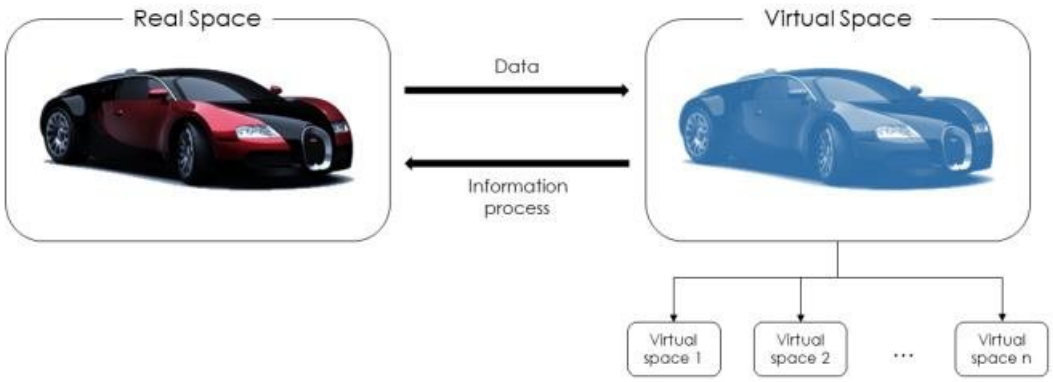

Figura 2.8: Modello di Digital Twin secondo la definizione di M. Grieves

2.3.1 Definizione

Il concetto di Digital Twin (DT) ha sempre avuto multiple definizioni. Non vi é un concetto chiaro e conciso che possa descrivere in maniera diretta cosa sia. Tale problema é dato dall'ampio ambiente applicativo in cui i Digital Twin vengono formalizzati (Aereonautica, Robotica, Manufacturing, Monitoraggio di Processi, ecc..). Formalmente, una prima definizione di Digital Twin é stata data da Michael Grieves nel 2002, all'interno della sua pubblicazione relativa alla Product Life-Cycle Management (PLM) col titolo "Conceptual Ideal for PLM" . In seguito, il concetto di DT di Grieves fu definito e applicato all'interno di altri paper [21, 22, 23] fino a formalizzarne un concetto formale di base utilizzato come riferimento per le applicazioni di ricerca negli anni successivi. Da ció, possiamo definire un Digital Twin come una componente formata da 3 elementi principali:

- Uno spazio reale in cui vengono definiti gli oggetti fisici
- Uno spazio virtuale in cui vengono implementati gli oggetti virtuali
- I link che definiscono il flusso di dati dallo spazio fisico allo spazio virtuale

Tramite tale composizione, un Digital Twin ha un flusso di dati dipendenti dallo spazio fisico che, comunicando tramite i link, riesce a prelevare i dati dello spazio reale e propagarli verso spazi e sottospazi virtuali, definendo un contesto virtuale completamente dipendente da quello reale. La figura 2.8 mostra una rappresentazione grafica delle interazioni tra le tre componenti

secondo il modello di M. Grieves. Partendo da questo concetto di base, l'evoluzione dei Digital Twin é stata portata sempre piú verso un concetto complesso; rendendolo, ad esempio, una delle soluzioni migliori per sistemi di diagnostica, andando ad essere utilizzati anche dalla U.S. Air Force [24], per la gestione di Cyber-Physic Systems (CPS), digital manufacturing, fino alle moderne tecnologie Industry 4.0 odierne.

2.3.2 Funzionamento

Partendo dal concetto esposto nella sezione precedente, un Digital Twin rappresenta un mezzo informativo prezioso per molte applicazioni di Intelligenza Artificiale (AI), Internet of Things (IoT) e Cloud. La combinazione di tali tecnologie applicate nell'analisi di sensori dello spazio reale, riescono ad analizzare gli aspetti delle funzionalitá di un oggetto. In altri termini, i dati prodotti dai sensori su diversi aspetti prestazionali di un oggetto fisico (es. temperatura, pressione, condizione atmosferica, umidita, ecc...), insieme ad un apprendimento del comportamento di una Twin, hanno come obiettivo finale quello di poter comprendere l'andamento dei dati, analizzare il comportamento dell'oggetto fisico e predirre come esso si comporterá in futuro. Questo flusso di informazioni continuo di dati reali, permette al Digital Twin di mantenere lo stato dell'oggetto reale in maniera continua nel tempo. Un Digital Twin permette anche l'esecuzione di simulazioni in modo da poter rilevare possibili problemi diagnostici ancor prima che essi accadano; cosi da poter correggere le funzionalitá dell'oggetto fisico connesso ed evitare errori o malfunzionamenti. Inoltre, le simulazioni su un Digital Twin vengono anche utilizzati da sistemi di analisi; in modo da poter studiare possibili miglioramenti del sistema. Nella figura 2.8, il flusso di comunicazione tra il prodotto fisico e il Digital Twin é bidirezionale: l'oggetto reale provvede a fornire i dati al Digital Twin per poter studiare, analizzare e predirre miglioramenti, il Digital Twin restituisce i risultati dell'analisi in modo da fornire un feedback sulle funzionalitá del sistema nello spazio reale.

2.3.3 Tipi di Digital Twin

Ci sono vari tipi di Digital Twin che dipendono dal livello di gestione rispetto al dispositivo fisico di riferimento. La piú grande differenza tra questi tipi é data dalle varie aree di applicazione. Nelle infrastrutture moderne,

piú tipi di Digital Twin coesistono all'interno di uno stesso sistema o di un medesimo processo. Ció sta a significare che si possono distinguere vari tipi di Digital Twin in dipendenza dei differenti campi di applicazione. Le maggiori differente si trovano sul livello di astrazione applicativo connesse al Twin. Tali astrazioni possono diversificare i seguenti tipi di DT:

1. **Components/Parts Twins:** Sono dei Digital Twin che rappresentano l'unitá di base logica. Essa é collegato direttamente ad una unitá meccanica di un dispositivo; che rappresenta, in maniera logica, una funzionalitá di base del macchinario. Generalmente la definizione di Parts Twins e Components Twins sono molto simili in letteratura. Peró, in alcuni casi, vengono definiti come due categorie differenti. Nello specifico, le Parts Twins rappresentano unitá ancor piú piccole delle singole Components Twins. In generale, tali tipi di Twins fanno riferimento all'analisi e allo studio delle singole funzionalitá meccaniche di un dispositivo e non si proiettano sul funzionamento a cui esse sono collegate.

2. **Asset Twins:** Quando vi sono due componenti o parti che lavorano insieme, si parla generalmente di asset. Gli Asset Twins consentono di studiare l'interazione tra le componenti, creandone una vasta gamma di informazioni sulle prestazioni in grado di poter essere utilizzate nell'elaborazione dato; per poi trasformate in informazioni fruibili per il dispositivo fisico o per processi di valutazione e di testing.

3. **System or Unit Twins:** Fanno riferimento ad un gruppo di assets che condividono gli stessi obiettivi logici nei processi del sistema. In altri termini, possiamo definire un System Twins come una entitá in grado di vedere il modo in cui diverse risorse si uniscono per formare un intero sistema funzionale. Esse forniscono visibilitá sull'interazione delle risorse e possono suggerire miglioramenti delle prestazioni all'interno dell'intero sistema.

4. **Process Twins:** Raffigurano dei Twin in grado di stabilire e rilevare come i sistemi interagiscono per creare un intero impianto di produzione. Questi sistemi sono tutti sincronizzati per funzionare alla massima efficienza o in ritardi controllati (nel caso di un processo di produzione in campi come il manufacturing). Nello specifico, visualizzano come le differenti operazioni tra i sistemi si influiscano l'un l'altro. Questi tipi

di Twin possono essere utilizzati per determinare gli schemi temporali precisi che possono influenzare l'efficienza complessitá di un processo.

2.3.4 Vantaggi nell'uso di Digital Twins

La grande diffusione dei Digital Twins in campi della Industry 4.0 e, in generale, nei sistemi intelligenti moderni, é portata sicuramente da alcuni benefici d'utilizzo che hanno dato un rapido miglioramento a differenti fattori:

1. **Miglioramenti in Ricerca e Sviluppo:** In campo R&D (Research and Development), l'uso dei Digital Twins consente una analisi e una progettazione di prodotti affidabile ed efficiente, con un abbondanza di dati creati sui probabili risultati delle prestazioni nei processi di sviluppo, testing e prototipizzazione. Tali informazioni possono portare ad approfondimenti che aiutano le aziende a perfezionare i prodotti da distribuire partendo giá dalla fase prototipale.

2. **Maggior efficienza:** Anche dopo essere entrati in fase di produzione, i Digital Twins forniscono uno strumento di supporto in grado di aiutare a rispecchiare e monitorare i sistemi di produzione, con l'obiettivo di raggiungere e mantenere la massima efficienza durante l'intero processo produttivo. Altri esempi vengono dati dalle procedure di decision-making, volti a definire un analisi sull'ottimalitá delle scelte operazionali svolte durante task di monitoraggio o di analisi dati.

3. **Gestione di Prodotti End-of-Life:** I Digital Twins possono essere utilizzati anche per definire un processo decisionale in grado di aiutare a definire quali opzioni un ingegnere dovrebbe seguire e cosa fare con i prodotti che raggiungono la fine del loro ciclo di vita. Nel pratico, essi possono ricavare informazioni su alcune delle funzionalitá o delle parti fisiche o logiche di un prodotto e, con i giusti metodi, poterle reciclare per riutilizzarle su altri dispositivi; cosi da mantenere una ottimalitá di gestione anche sotto un punto di vista di risparmio di costi data dallo smaltimento di prodotti obsoleti.

2.3.5 Differenze tra Digital Twin e Simulazioni

Una simulazione raffigura un processo in grado di creare un modello digitale in grado di imitare la realtá al meglio delle informazioni raccolte sullo

spazio fisico, riproducendo gli eventi, e definire un processo del sistema che reagisce agli input esterni; in maniera fedele rispetto al comportamento di dispositivi reali simulati. Le simulazioni sono diventate negli anni uno dei punti fondamentali per vari campi, specialmente in settori come quello di produzione. Questo perché, tramite un processo di simulazione, si é in grado di analizzare tutte le fasi di assemblaggio o di creazione di un prodotto, mantenedo uno stato di supervisione anche in ambienti legati alla supply chain di un industria. Tale processo, peró, risulta essere statico e preconfigurato; ossia affronta casi solamente in un ambiente ideale, senza reagire ad eventi non previsti o all'aggiunta di nuovi parametri che possano influenzare lo stato del sistema. Come specificato nelle sezioni precedenti, un Digital Twin rappresenta anch'esso un modello digitale di un processo fisico, ma diversamente dalle simulazioni, il suo comportamento é dinamico e rispecchia la realtá grazie ad un canale di comunicazione bidirezionale con il sistema fisico in grado di garantire una dipendenza sui dati reali in real-time. Anche se si conosce a pieno un sistema, un simulatore non avrá mai la garanzia di gestire tutti i tipi di dati del contesto reale come, invece, avviene per i Digital Twin. Da ció, possiamo vedere i Digital Twin come una estensione delle simulazioni; in cui, non sono solo in grado di analizzare alcune caratteristiche del sistema, ma riescono a ricavare dati direttamente nel contesto reale, anche con tempi molto brevi. Inoltre, grazie ai Digital Twin, si ha la possibilitá di intervenire tempestivamente sul sistema reale in processi legati all'efficienza della progettazione, ottimizzazione di processi, correzione di errori e altre procedure simili che, al contrario, non possono essere tempestivamente eseguite tramite le simulazioni.

2.3.6 Esempi di Applicazioni

Negli anni, i Digital Twin si sono trasformati da un mezzo di gestione di singoli asset ad un meccanismo di gestione di sistemi complessi come, ad esempio, le smart building. In questa sezione andremo ad analizzare alcuni esempi di applicazione dei Digital Twin che hanno rivoluzionato le applicazioni in molti campi dell'Industrial IoT e dell'Industry 4.0.

Digital Manufacturing All'interno del Manufacturing industriale, i Digital Twin sono risultati vantaggiosi non solo sotto il punto di vista di analisi dei dati sullo stato reale del sistema di produzione, riproducendone dinami-

che con accuratezza; ma vengono anche maggiormente utilizzati per prevedere gli stati futuri di un sistema partendo dallo stato e i dati del presente. Inoltre, esso viene utilizzato anche per garantire la fattibilitá e l'efficienza di una soluzione complessa, anche se si trova ancora in fase di progettazione. Questo perché, nelle fasi di progetto preliminari di un impianto di produzione, l'applicazione delle Digital Twin permette di definire un modello di simulazione accurato, in modo da consentire al progettista di ottenere dati precisi sulle prestazioni di un sistema e adottare, ancor prima della creazione o dell'utilizzo effettivo, scelte progettuali che possano avere una prospettiva chiara e certa sulle prestazioni e l'efficienza di un impianto. Successivamente, quando lo spazio virtuale del Digital Twin si interfaccia verso il sistema di produzione reale, l'acquisizione di dati di produzione, fornita dai sensori dell'impianto, permette di definire criticitá o errori nel processo produttivo. Tale meccanismo viene comunemente utilizzato per evitare che una catena di produzione abbia degli errori di fabbricazione di massa, andando a rilevare criticitá di un possibile macchinario tramite l'analisi dei suoi sensori e, mediante la generazione di alert del Digital Twin, poter adoperare meccanismi di mitigazione o di risoluzione del problema hardware o software del sistema. Infine, uno degli aspetti critici dei DT industriali é la predisposizione dell'interfaccia verso l'impianto. Il Digital Twin deve essere conforme in ogni aspetto semantico dei macchinari fisici di produzione, in modo da poter raccogliere tutte le informazioni necessarie per poter adoperare una corretta analisi dei dati, definire soluzioni ottimali e predirre possibili stati di malfunzionamento. [25].

Automotive Il campo dell'automotive é cambiato drasticamente negli anni. I consumatori tendono ad acquistare sempre auto innovative, con sistemi e tecnologie integrati intelligenti, capaci di fornire servizi aggiuntivi al loro acquisto. Le compagnie automobilistiche hanno adottato un piano di produzione incentrato sempre piú sull'utilizzo e l'integrazione di sistemi digitali con tecnologie emergenti, basate sull'utilizzo ausiliario di sensori e di altri dispositivi in grado di migliorare l'esperienza e la comoditá del veicolo. I Digital Twin ricoprono una parte importante all'interno di tutto il ciclo di vita del veicolo, dall'incremento delle performance e dall'ottimizzazione del design in fase progettuale, in modi simili al manufacturing, rendendo i modelli sempre piú performanti e con prestazioni ottimizzate giá dalla fase prototipale. I Digital Twin hanno incrementato anche il modus operandi delle tecnolo-

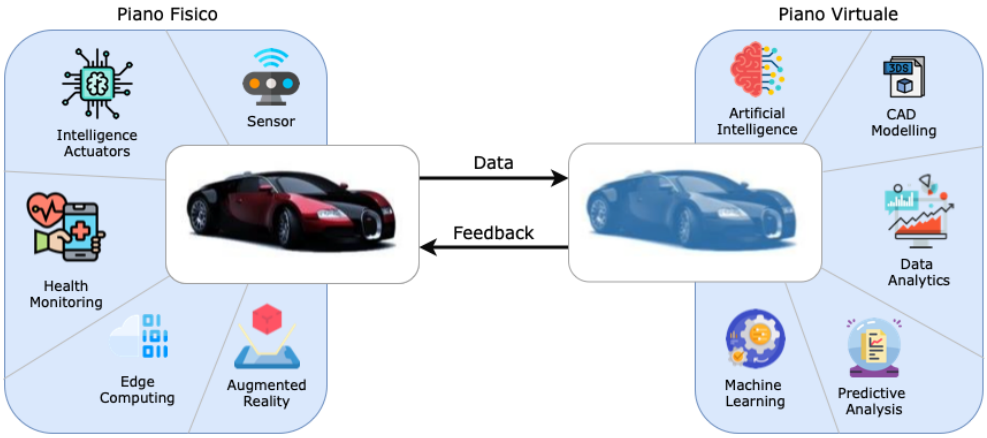

Figura 2.9: Applicazioni dei Digital Twin nel campo dell'Automotive Industriale

gie connesse al veicolo, ottenendo un controllo real-time dello stato dell'auto con un DT connesso, in modo da mantenere un registro digitale constantemente aggiornato. Molte aziende automobilistiche impiegano i Digital Twin all'interno del loro processo produttivo per fornire servizi aggiuntivi ai loro veicoli. Un esempio é dato da Tesla Motors, casa automobilistica americana, in cui utilizzano i Digital Twin sviluppati da Thinkwik[2], un azienda partner nota nel campo automotive e IoT, in grado di rilevare in tempo reale i problemi di una vettura e, indipendente dall'entitá del problema, risolverlo semplicemente scaricando aggiornamenti software over-the-air (OTA)[26].

Smart Building Quando si parla di Digital Twins si fa sempre riferimento ai singoli asset che possono essere logicamente gestiti da una copia nello spazio virtuale. Questa caratteristica, peró, non é l'unica soluzione possibile per l'applicazione dei DT a sensori o dispositivi fisici. Infatti, nelle infrastrutture IoT moderne, si é sempre piú parlato di Digital Twin formati da aggregati di dispositivi che lavorano insieme per raggiungere un obiettivo comune. Le Smart Building sono uno degli esempi piú chiari di queste applicazioni, definiscono uno stato globale su ogni edificio, andando ad astrarre caratteristiche legate ai singoli sensori, mostrando pattern e dati interscambiati all'interno del sottosistema (che comprende tutti i sensori o dispositivi di un edificio). Questo porta alla gestione di un livello di astrazione piú ampio, capace di po-

[2]https://www.thinkwik.com

ter definire informazioni che possano dare ulteriori servizi al sistema IoT su cui sono basati. Esempi di questo genere si sono visti con l'applicazioni IoT di gestione di sistemi energetici, in grado di stabilire la temperatura media di un edificio, definendone i consumi e ottimizzando l'utilizzo di apparecchi di termoregolazione definendo obiettivi di risparmio energetico. Altri esempi in letteratura, possono essere riferiti a tutti quei casi di Smart Home in cui il piano energetico é automatizzato in relazione all'energia prodotta da pannelli fotovoltaici, pale eoliche o altre fonti di produzione di energia domestica che possono essere direttamente correlati alla gestione dei consumi domestici. [27]

Sanitá Quando si parla di Digital Twins si fa sempre riferimento ad asset che rappresentano prodotti. I contesti piú discussi fanno riferimento all'analisi di produzione, al monitoraggio di sistemi e all'ottimizzazione di processi tipicamente applicati in ambienti industriali. Con l'avvento della pandemia e la diffusione del COVID-19, i Digital Twins hanno avuto una rapida diffusione anche in campo sanitario. Nello specifico, si é parlato di definire dei sistemi in grado di implementare una versione digitale di un paziente basandosi su dati di analisi, condizioni cliniche e altre informazioni che portano ad ottenere uno screening di una persona, andando a definire possibili pattern in grado di evidenziare problemi di salute; cosi da fornire un sistema di supporto ai medici durante le diagnosi. [28]

Estendibilitá in Applicazioni Future Con il continuo espandersi dei domini di applicazioni dei Digital Twin e con il continuo incremento della complessitá dei sistemi, vi sará, nei prossimi anni, un continuo aumentare dell'utilizzo di tali tecnologie. Questi eventi hanno portato ad un cambiamento fondamentale rispetto ai modelli operativi esistenti. Una reinvenzione digitale si sta verificando nei settori ad alta intensitá di risorse; richiedendo una visione fisica e digitale integrata sulle componenti, sulle apparecchiature, sulle strutture e sui processi. I Digital Twin ricoprono e ricopriranno un ruolo fondamentale per tale riadattamento. Con l'aumentare della potenza computazionale messa a disposizione nei sistemi della Industry 4.0, i Digital Twin esistenti hanno anche il potenziale di poter essere estesi semanticamente, andando ad integrare nuove funzionalitá, capacitá e competenze. Ció significa che possono continuare a generare le informazioni necessarie per migliorare i prodotti e rendere i processi sempre piú ottimali in maniera indipendente

dall'aumentare della complessitá di un sistema che, se adeguatamente gestita tramite risorse computazionali proporzionate, possono ottenere importanti risultati per il miglioramento dei processi.

Capitolo 3

Progettazione

3.1 Obiettivi

L'obiettivo del progetto é legato alla sperimentazione di un meccanismo di generazione di Web Digital Twin in grado di poter simulare il comportamento di una Web Thing fisica o di una sua simulata. Nello specifico, lo scopo primario avrá come fine quello di poter studiare quali possano essere i vincoli o le difficoltá nella generazione di un Web Digital Twin efficiente. In particolare, andremo a sviluppare un sistema di generazione di Web Digital Twin capace di poter prendere modelli espressi matematicamente all'interno della Thing Description (secondo una definizine fornita dai provider dei dispositivi o dai programmatori dell'ambiente IoT). Inoltre, analizzeremo l'ottimalitá dei parametri uscenti dalla fase di addestramento e applicheremo il modello finale ad un modulo di generazione, capace di poter definire un Digital Twin che possa comportarsi in maniera simile alla Web Thing in input; minimizzandone le differenze tra i valori di stato dello spazio reale e virtuale.

3.2 Requisiti Funzionali

I requisiti funzionali si interfacciano verso una descrizione ad alto livello dei servizi che si implementeranno nel sistema proposto. Nello specifico, rappresentano tutte quelle funzioni di base capaci di poter essere eseguite, direttamente o indirettamente, per poter cambiare lo stato del sistema. Di seguito, elenchiamo i requisiti funzionali integrati in fase progettuale:

1. Implementazione di un dispositivo IoT in grado di fornire informazioni sullo spazio reale attraverso sensori e attuatori simulati.

2. Interfacciamento verso lo standard W3C WoT tramite la creazione della Thing Description per le specifiche sulle interaction affordance.

3. Applicazone delle Scripting API per l'implementazione delle Interaction Affordance descritte nel TD.

4. Monitoraggio della Web Thing e raccolta dati sulle osservazioni di stato e azioni per un tempo $t \in \Gamma_T = [t_0, t_n]$ con $t_i - t_{i-1}$ variabile per $i = 1...n$.

5. Parsing dei modelli logici della Thing Description per definire parametri P ottimali che diminuiscano la devianza tra i valori osservati e predetti tramite un processo di addestramento.

6. Generazione Web Digital Twin basato sul modello ottimale ottenuto nel punto precedente.

I requisiti funzionali verranno implementati ed utilizzati in maniera modulare in uno o piú servizi dell'architettura.

3.3 Modello Comportamentale di una Web Thing

All'interno di questa sezione, descriveremo le caratteristiche del modello comportamentale su cui si interfaccerá la logica semantica di distribuzione delle proprietá di un Web Digital Twin. In genere, chi distribuisce o gestisce i sensori, conosce le caratteristiche delle misurazioni sui vari tipi di dati che un dispositivo acquisisce. Da ció, é possibile definire un modello matematico in cui si mettono in luce le possibili dipendenze del sistema. In altri termini, si definisce matematicamente la variazione di una proprietá $p \in P$ per un determinato tempo $t \geq 0, t \in \mathbb{N}$. Il comportamento di un dispositivo o di una serie di dispositivi, peró, é generalmente influenzato da fattori esterni che non possono essere controllati e predetti ma sono definiti dall'ambiente in cui una Thing é immersa. Per questo motivo, i modelli introducono delle variabili parametriche con lo scopo di modificare la distribuzione dei valori nel tempo; basandosi su osservazioni delle variabili di stato (proprietá) di una Web Thing. Partendo da tali osservazioni, il modello puó essere matematicamente riassunto nel seguente modo:

Dato un sistema IoT con $d_1 ... d_n \in D$ dispositivi tali che $\forall d_i \in D$, aventi variabili $S_i = s_{i,1}, s_{i,2} ..., s_{i,n} = R_i \cup W_i$ dipendenti dal tempo t, in cui gli insiemi R_i e W_i sono definiti come:

1. R_i l'insieme delle proprietá *read-only*, ossia valori che possono solamente essere letti dal dispositivo d_i associato
2. W_i l'insieme delle proprietá *write-only*, ossia variabili che possono solamente essere modificate su un dispositivo d_i associato attraverso l'esecuzione di azioni.

Possiamo descrivere l'intero sistema in una singola Thing Description in grado di determinare, in dipendenza del tempo:

1. $R = \cup_i R_i$ l'insieme delle proprietá *read-only* del sistema
2. $W = \cup_i W_i$ l'insieme delle proprietá *write-only* del sistema.
3. $S = W \cup R$ l'insieme dei valori delle proprietá di un sistema

Da ció, possiamo definire due tipi di equazioni per le proprietá $r_k \in R$:

1. **Equazioni Algebriche:** In grado di definire matematicamente il valore di una proprietá ad un determinato istante temporale.

2. **Equazioni Differenziali:** In grado di descrivere l'evoluzione dei valori di una proprietá al variare del tempo.

Dato che ogni proprietá definita in una Thing Description di un sistema IoT puó essere espressa tramite solo da una funzione algebrica o differenziale, possiamo dividere le proprietá di R in due sottoinsiemi $A, B \subset R$ in cui:

1. $\forall a_j \in A$ si definisce una funzione algebrica g_j:

$$a_j(t) = g_j(t, B(t), W(t), P_j)$$

2. $\forall b_j \in B$ si definisce una funzione differenziale f_j:

$$\dot{b}_j = f_j(t, A(t), B(t), W(t), P_j)$$

In cui $A(t)$, $B(t) \subseteq R(t)$ e $W(t)$ raffigurano le variabili su cui $a_j \in A$ o $b_j \in B$ dipendono funzionalmente, ossia quelle proprietá che, al variare del loro valore, influenzano lo stato della proprietá che si sta calcolando. P_j raffigurano i coefficienti parametrici di una proprietá $j \in R$, essi possono avere dei valori iniziali forniti dal distributore del modello. Essi possono far riferimento ad uno scope locale, legato al singolo modello, o globale, legato alla Thing Description. Nel secondo caso, il parametro puó essere condiviso in piú proprietá di R. In altri termini, date le proprietá $r_i \in R$, l'insieme delle variabili globali é definita come $K = \{K_j | K_j = \bigcup_{r_j, r_i \in R} P_j \cap P_i\}$. Nella fase di learning, tali coefficienti verranno modificati in modo da essere associati a valori ottimali che descrivano il comportamento di una Web Thing monitorata o simulata. In altri termini, partendo da funzioni di base:

$$G = \{a_j(t) = g_j(t, B(t), W(t), P_j) \land a_j \in A \land t \in \mathbb{N}\}$$

$$F = \{b_j = f_j(t, A(t), B(t), W(t), P_j) \land b_j \in B \land t \in \mathbb{N}\}$$

possiamo modificare i parametri P_j in modo da ottenere funzioni approssimate di G e F, definite come $\hat{G}$ e $\hat{F}$, con valori ottimali per descrivere il comportamento delle proprietá dei dispositivi $d_j \in D$, $j = 1...n$ e predirre i valori futuri Z, con Z l'insieme delle variabili predicibili ($Z \subseteq R$), per un tempo $t' \geq t, t' \in \Gamma_Q, t \in \Gamma_T$ e $\Gamma_T \subseteq \Gamma_Q$, con Γ_Q definito come l'insieme dei

valori temporali . Nella prossima sezione, andremo a descrivere i campi della Thing Description associati alle varie componenti del modello comportamentale con la relativa sintassi. Mentre nella sezione 3.5, si analizzerá l'utilizzo dei campi introdotti per definire una struttura logica ottimale.

3.4 Estensione Web of Things

Nella sezione 2.2 abbiamo introdotto i concetti di Thing Description e di Web Thing. Le informazioni contenute all'interno di una Thing Description fanno riferimento ad alcuni metadati che sono importanti per poter garantire la comunicazione in maniera interoperabili tra piú dispositivi IoT. Lo standard fornito da W3C non mette in luce alcune informazioni che possono essere utile per l'analisi del comportamento di una Thing fisica. Per determinarne un modello del comportamento secondo la definizione discussa in 3.3, sono stati introdotti tre differenti tipi di campi semantici:

1. **model:** Raffigura una espressione seguendo la sintassi Python formata da quattro parti principali:
   (a) **Comportamento:** Puó assumere il valore di $self$ nel caso si faccia riferimento ad una espressione algebrica in cui il valore é determinato in relazione all'istante di tempo t, oppure $dot(self)$ nel caso in cui si ha una funzione differenziale in grado di rappresentare la variazione dello stato di una proprietá al variare del tempo.
   (b) **Funzione:** Raffigura una espressione matematica in grado di rappresentare una combinazione lineare in grado di definire la variazione di uno stato nel tempo o lo stato effettivo in uno specifico istante t dipendente dal tipo di comportamento a cui é associato. La sua sintassi segue le regole grammaticali di Python per la stesura delle espressioni. Nello specifico, mette in relazione aritmetica varie componenti:
      i. Un insieme di parametri locali definiti da $params[i]$ con i l'indice del parametro a cui si fa riferimento. Tale indice é condiviso tra tutti i parametri locali della Thing Description
      ii. Un insieme di parametri globali definiti dalla collezione $globals[j]$ con j il numero di parametri globali presenti nella Thing De-

scription. Esso fa riferimento a quei valori condivisi tra piú modelli della TD.

iii. Proprietá di input in grado di poter far riferimento al valore di proprietá linearmente dipendente specificata dalla clausola $input(nome)$ oppure ad un insieme di proprietá di un particolare tipo semantico tramite $inputType(tipo)$. In entrambi i casi, i valori a cui si fanno riferimento devono essere specificati all'interno del campo $modelInput$, definita successivamente, in modo da associare i valori del *nome* o del *tipo* correttamente.

(c) **Vincoli:** Raffigura un espressione che definisce un range di valori che un valore in *params* e *global* puó ottenere durante il processo di ottimizzazione. Inoltre, dato che *global* é definito nello scope della Thing Description, i suoi vincoli possono essere specificati solamente una volta per tutti i modelli presenti nel documento. Dato che ogni parametro é racchiuso all'interno di un range che ne specifica i valori possibili che puó assumere, nel caso nessun vincolo faccia riferimento all'elemento minore o maggiore, essi saranno automaticamente impostati a $-\infty$ e $+\infty$. Una forma generica che rispetta le condizioni sintattiche puó essere:

$$params[i] <= v_i, params[j] >= v_j, globals[k] <= v_k$$

$$\text{con } v_i,\ v_j,\ v_k \in \mathbb{R},\ i, j, k \geq 0$$

(d) **Parametri Iniziali (Guess):** Raffigura i valori iniziali per le collezioni *params* e *globals* prima del processo di ottimizzazione all'interno della fase di apprendimento. Come per i vincoli, i valori di *globals* devono essere definiti una volta per tutti i modelli della Thing Description. Mentre i vincoli raffiguravano condizioni, le espressioni di guess rappresentano assegnazioni. Un formato generico che rispetti le regole sintattiche puó essere come segue:

$$params[i] = v_i, params[j] = v_j, globals[k] = v_k$$

$$\text{con } v_i,\ v_j,\ v_k \in \mathbb{R}$$

In conclusione, possiamo definire un modello come la concatenazione delle componenti precedentemente descritte:

$$Comportamento = Funzione \mid Vincoli \mid Guess$$

Tale espressione andrá in input al Learning Module e verrá processato da un parser in modo da modificarne la forma e rendere le informazioni accessibili all'optimizer.

2. **valueFrom:** Identifica dove verranno letti i valori della proprietá, puó assumere due valori:
   - *readProperty* nel caso il dato della proprietá faccia riferimento alla lettura su un sensore reale.
   - *model* nel caso il dato é uscente dal modello, ossia é simulato in relazione alla valutazione algebrica o all'integrazione della funzione del modello.
3. **modelInput:** Raffigura la lista delle proprietá utilizzate in un modello tramite *input*() o *inputType*() definite nella sintassi del *model*. Nello specifico, raccoglie tutti quei parametri utilizzati nel modello che fanno riferimento ad una proprietá dipendente utilizzata. Nello specifico, ogni proprietá é espressa secondo i seguenti campi:
   (a) *title* che specifica il riferimento della proprietá all'interno del modello preso sotto analisi.
   (b) *propertyName* definisce il nome della proprietá della Thing Description legata all'input.
   (c) *type* specifica il tipo dei valori dell'input
   (d) *model* che definisce un modello secondario proprio dell'input da dover importare all'interno dell'espressione principale.
   (e) *modelType* rappresenta un campo facoltativo che accoppia semanticamente uno o piú input tra loro. Esso specifica il riferimento per il parametro della clausola *inputType*() all'interno dell'espressione di *model*.

Tramite le informazioni dell'estensione, si puó definire un modello comportamentale basandosi sull'ottimizzazione dei parametri dati in input, in modo da poter adattare il modello generico agli eventi e alle variazioni di stato date dall'analisi della Web Thing immersa all'interno dell'ambiente reale o simulata in uno spazio virtuale.

3.5 Processo di Learning

Descriveremo brevemente il processo di learning definito dentro il modulo dedicato, analizzando il procedimento di ottimizzazioni dei parametri del modello su cui si baseranno i Web Digital Twin generati.

3.5.1 Parsing del Modello

Una volta ricevuto in input i dati o il file contenente il tracciamento dei cambiamenti di stato di una Web Thing, bisogna definire matematicamente i modelli contenuti all'interno della Thing Description; seguendo la grammatica Python per la loro esecuzione. Nel processo di parsing, vengono adottate le seguenti regole:

1. Si fa distinzione tra funzioni differenziali $f_i \in F$ e quelle algebriche $g_k \in G$ tramite, rispettivamente, $dot(self)$ e $self$ del modello dato in input; sostituendoli rispettivamente con $y[i]$ e $dxdt[k]$.

2. L'espressione $value()$ che definisce la lettura da un sensore fisico viene sostituita con la funzione $readProperty(P, t, d)$ con P il nome della proprietá in $R(t)$, t il tempo di scansionamento e d i dati temporanei.

3.5.2 Learning

Il processo di learning ha lo scopo di definire i parametri $\hat{P}$ tali che possano delineare un modello comportamentale significativamente piú simile rispetto al comportamento di una Web Thing. Il processo di learning ha i seguenti valori in input:

1. P_0, ossia i valori delle proprietá definiti secondo i parametri di guess all'interno del dominio espresso nei vincoli.

2. $O = (O_{t_0}, O_{t_1}, ..O_{t_n})$, una sequenza di osservazioni nel dominio temporale $\Gamma_t = [t_0, t_n] \subset \Gamma$ con Γ l'insieme continuo dei tempi di osservazione.

Partendo da una funzione h tale che $O_t = h(t, B(t), A(t), W(t), \hat{P})$, ossia che rappresenti un modello di osservazione che, al variare dei valori di $\hat{P}$ puó migliorare o peggiorare la differenza tra il valore osservato e quello predetto dal modello parametrico. L'algoritmo di learning si basa sulla seguente minimizzazione:

$$minimize_{P\in\mathbb{P},R(t_0)\in\mathbb{P}}\sum_{k=0}^{n}||H_{W_{\Gamma_N},\hat{P},R(t_0)}(t_k)-O_k||^2$$

In cui $H_{W_{\Gamma_N},\hat{P},R(t_0)}(t) := h(t, B_{W_{\Gamma_T},\hat{P},R(t_0)}(t), W_{\Gamma_Q}(t), \hat{P})$ é una funzione definita in relazione ad un tempo $t \in \Gamma$, con l'esecuzione di azioni $W_{\Gamma_Q}(t)$ con Γ_Q tale che $\Gamma_T \subset \Gamma Q$ con $\Gamma_Q = [t_0, t_q]$ e $t_q \geq t_n$ su osservazioni dipendenti dal tempo O_k, $k = 1..., n$. I valori parametrici P e i valori iniziali $R(t_0)$ sono definiti in un dominio superiormente e inferiormente limitato $\mathbb{P}$. Il procedimento di ottimizzazione che parte da P e $R(t_0)$ segue, mantenendo i vincoli dei limiti in $\mathbb{P}$, l'algoritmo non lineare di Least Squares in grado di definire i valori successivi per $\hat{P}$ e $\hat{R}(t_0)$ ottimali.

3.5.3 Inferenza

Definendo i parametri ottimali $\hat{P}$ e $\hat{R}(t_0)$, un intervallo di tempo $\Gamma_T = [t_0, t_n]$, un insieme di azioni sulle proprietá W sull'intervallo Γ_T (W_{Γ_T}), possiamo determinare la storia degli stati dei valori monitorati da una Web Thing tramite l'applicazione delle funzioni differenziali F e algebriche G del modello parametrico. Per fare ció, si applica una equazione differenziale ordinaria non autonoma e parametrica (ODE) basata per ogni azione eseguita, definendola per ogni proprietá $b_j \in B$, in dipendenza dei parametri $\hat{P}$. Nello specifico, essa é definita secondo la formula:

$$b_j(t) = f_j(t, B(t), A(t), W_{\Gamma_T}(t), \hat{P}_j) \text{ con } t \in \Gamma_T$$
$$r_j(t_0) = \hat{r_j}(t_0)$$

Risolvendo l'equazione, possiamo determinare i valori $\hat{b_j}(t)$ e usarli per definire una soluzione alle funzioni algebriche g_i associate alle proprietá a_j con tempo t, valori iniziali $\hat{R}(t)$ e parametri $\hat{P}_i$.

3.6 Generazione Web Digital Twin

I parametri ottimizzati del learning module verranno utilizzati per definire l'inferenza sulle predizioni dei valori futuri, in relazione alla variazione dei dati su cui sono stati allenati. In altri termini, partendo dal modello addestrato, si andrá a creare un Twin module che simulerá il comportamento della Web Thing fisica, in relazione ai dati osservati e alle azioni eseguite. Per poter effettuare tale operazione, si parte da una versione modificata della

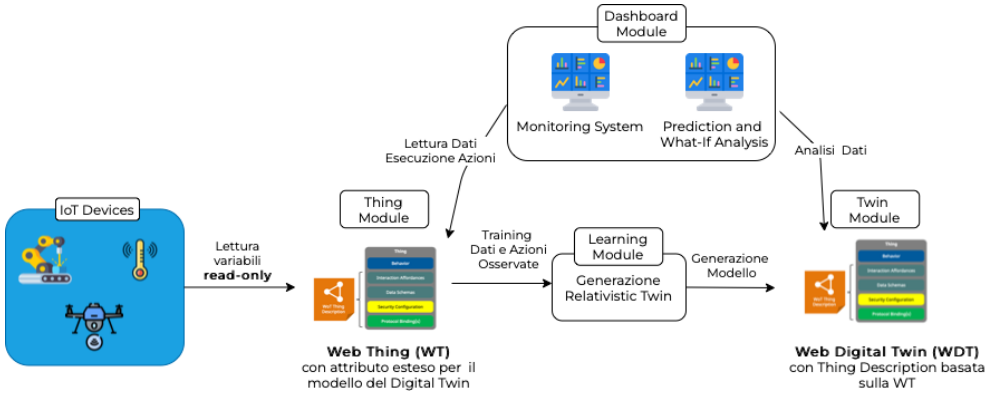

Figura 3.1: Flusso Dati per la generazione e l'iterazione di Web Thing e Web Digital Twin all'interno dell'architettura proposta.

Thing Description, in cui, per tutte le proprietá espresse dalla Web Thing fisica, se ne genererá una rispettiva, dipendente dal tempo, che, dato in input, stabilirá il valore della misurazione di una proprietá per un particolare istante temporale. Dalla Thing Description generata, si avrá la possibilitá di definire un interfaccia verso un oggetto Twin in grado di effettuare una iterazione simile all'utilizzo dell'interfaccia della Web Thing, basandosi sulle osservazione dei dati reali. Partendo da tale architettura, un servizio esterno potrá comunicare con una Web Thing o un Relativistic Digital Twin utilizzando, semplicemente, le interfacce messe a disposizione da tali componenti. Nel sistema proposto dal framework, si utilizza una dashboard di monitoraggio per poter leggere lo stato delle proprietá ed eseguire azioni sulla Web Thing in una fase di raccolta dati. Successivamente si utilizzano tali dati per la generazione del modello con i parametri ottimizzati per la generazione del Web Digital Twin. Infine, si utilizza il Web Digital Twin per analizzare l'andamento dei dati, definire una what-if analysis sulla variazione dei valori e predirre valori su un intervallo futuro. Tali servizi, possono essere estesi verso altri casi d'uso in dipendenza del caso d'uso in cui il framework verrá utilizzato. In Figura 3.1 é presentata una versione grafica del flusso dati all'interno del framework, a partire dalla lettura dati dai dispositivi fisici.

3.7 Architettura

In questa sezione andremo a definire un illustrazione dell'architettura del progetto. Essa si basa su un design pattern a micro-servizi in cui i vari moduli trasferiscono e gestiscono i dati tramite canali di comunicazioni ben definiti. La figura 3.2 illustra la struttura principale del progetto, andando ad elencare i vari servizi e come essi comunicano tra loro. Nelle sottosezioni successive andremo a parlare piú nel dettaglio di ogni singolo modulo.

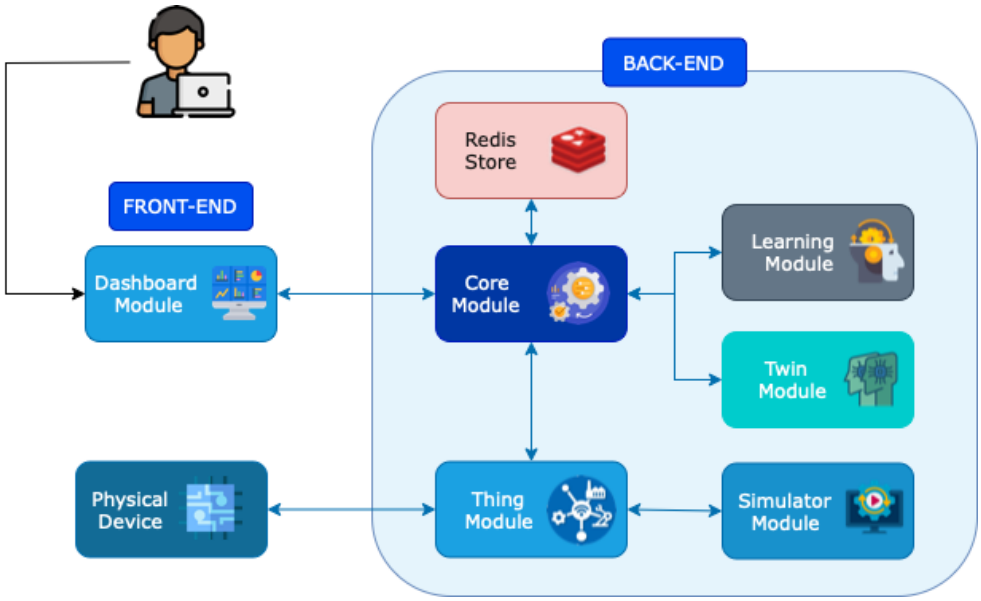

Figura 3.2: Architettura del Progetto

Il progetto utilizza come baseline l'architettura proposta in [29] andando a ristrutturare alcuni servizi, integrare nuove funzionalitá e rendere i servizi interoperabili e modulari.

3.7.1 Front-End

Dashboard Module

Il Dashboard module é diviso in due categorie di interazione:

1. Gestione e comunicazione con le Web Thing del sistema, andando a definire una fase di campionamento in cui poter interagire con la Web Thing tramite invocazioni di azioni o lettura delle proprietá, ad intervalli di tempo variabili.

2. Analisi e interazione con le Web Digital Twin, gestendone le azioni in dipendenza di un tempo dato in input, analizzandone il cambiamento di stato in relazione ad un azione eseguita.

Essa provvede a fornire un interfacciamento verso i servizi del sistema. Inoltre, fornisce strumenti in grado di:

1. Effettuare una simulazione su un ambiente virtuale, attraverso alcuni parametri definiti all'interno della GUI.

2. Definire una fase di monitoraggio dei dispositivi legati alla Web Thing, attraverso l'interazione indiretta con tale componente.

3. Invocare un azione della Thing Description, per variare lo stato della Web Thing in relazione ai dati di input inseriti. Tali azioni vengono vincolate al tempo logico della Web Thing, dipendente dal tempo di campionamento all'interno di una fase di monitoraggio.

4. Caricare file di tracciamento, in grado di poter riprendere sessioni di monitoraggio passate e addestrare dei modelli logici per la generazione di un WDT.

I dati di campionamento vengono salvati all'interno della sessione del Core Module e, attraverso la dashboard, si ha la possibilitá di poterli prelevare e salvarli in un file csv[1]. La struttura logica del file supporta il formato di input del modello del Learning module; ossia, si ha la possibilitá di sfruttarlo per la generazione di un nuovo Web Digital Twin. Quindi, possiamo definire tale interfaccia come uno strumento di gestione remota verso vari dispositivi. Da ció, possiamo affermare che il framework sfrutta al meglio l'interfacciamento web alla base del Web of Things. Oltre alla gestione per l'interazione con le WT reali, a partire dall'addestramento di almeno un modello, si puó interagire con una ulteriore interfaccia web. Essa é una dashboard in grado di definisce meccanismi di gestione delle Web Digital Twin generate. Poiché il framework puó interagire, mantenere e variare lo stato di una o piú WDT, l'utente finale puó sfruttare tale dashboard per differenti operazioni:

1. Selezionare e filtrare i tipi di WDT del sistema a secondo del tipo di Thing Description associata.

[1]Comma-separate value, formato dati tabellare in cui i valori delle colonne sono separati da una virgola e le righe da un simbolo new line.

2. Poter eseguire azioni non vincolate dal tempo; ossia, si ha la possibilitá di poter inserire o cancellare un azione in un istante $t \in \Gamma_t$.

3. Visualizzare la lista delle azioni legate ad uno dei WDT selezionati.

4. Definire un grafico di predizione, in grado di visualizzare l'andamento dei valori predetti dal modello del WDT.

5. Visualizzare l'andamento dei valori di una o piú WDT su uno su piú grafici, in modo da poter confrontare la precisione in relazione ai dati osservati.

6. Visualizzare un diagramma di stato; in cui i nodi e gli archi sono definiti come segue:

   (a) **Nodi:** Rappresentano lo stato del WDT ad un istante di tempo $t \in \Gamma_q$ con $\Gamma_q = \{t_0, t_1, ..., t_q\}$ un arco di tempo definito da un limite superiore e da una frequenza di scansionamento dati dall'utente.

   (b) **Archi:** Rappresentano le varie operazioni eseguite durante la transazione di stato. Possono essere azioni o semplici letture di proprietá.

7. Definire dei threshold che stabiliscono un limite inferiore e/o superiore in relazione ai valori dei nodi del diagramma di stato. Se uno dei vincoli di almeno una delle proprietá di stato S viene violato, il nodo assumerá un colore in base al tipo di evento catturato:

   (a) **Viola:** Se il valore di almeno una proprietá $p_j \in S_i$ é minore del suo vincolo inferiore.

   (b) **Rosso:** Se il valore di almeno una proprietá $p_j \in S_i$ é maggiore del vincolo superiore.

   (c) *Nero:* Se il valore di un sottoinsieme di proprietá $p_j \in S_i$ viola i vincoli inferiori e proprietá $p_z! = p_j \in S_i$ violano i vincoli superiori.

   (d) **Blu:** Se le proprietá di uno stato rispetta tutti i vincoli.

   Inoltre, si ha la possibilitá di poter definire una versione compatta del diagramma di stato, in cui le transizioni da uno stato S_i ad uno $S_(i+1)$

sono definite in relazione al cambiamento di stato in dipendenza dei threshold dati in input e dalle azioni eseguite. Tale strumento garantisce un meccanismo corretto e completo per poter definire operazioni legate ad un analisi what-if sui WDT prodotti.

In conclusione, possiamo affermare che il Dashboard Module definisce uno strumento completo per la gestione delle Web Thing e delle Web Digital Twin prodotte nel framework; andando a seguire il ciclo di vita di una WDT dalla fase di raccolta dei dati di addestramento, fino all'interazione e all'analisi dei suoi comportamenti.

Dispositivo Fisico

Rappresentano dei dispositivi fisici esterni al framework e fortemente vincolati al caso d'uso implementato. Sono raggiungibili attraverso il Protocol Binding delle WoT Scripting API eseguite in un WoT runtime dentro il Thing Module. Tali interfacce sono prodotte dai WoT Servient in back-end durante la fase di deployment dei moduli. Tramite tale canale di comunicazione, si ha la possibilitá di interagire con le varie componenti del sistema. Le Thing possono assumere qualsiasi configurazione, protocolli e formato dati, che vengono opportunatamente processate e adattate in dipendenza delle WoT Scripting API delle Thing Module, che rispecchiano la struttura logica definita all'interno della Thing Description associata. La comunicazione tra Web Thing e il dispositivo o i dispositivi fisici viene definita all'interno di una file di configurazione, in cui il WoT Servient acquisisce informazioni su quali protocol binding effettuare, quali specifiche adottare per i subprotocolli, se vi sono meccanismi di sicurezza e altri dati utili per poter implementare l'astrazione sulla comunicazione tra tali componenti. Tale modulo non verrá utilizzato nella validazione del framework, poiché il caso d'uso si baserá su soli oggetti simulati data l'impossibilitá di reperire attuatori capaci di essere utilizzati in un sistema IoT di esempio.

3.7.2 Back-end

Core Module

Il Core Module é uno dei moduli principali del back-end. Esso si interessa di gestire tutto il flusso di dati proveniente dagli altri servizi dell'architettura. Nello specifico, esso é responsabile di:

1. Instaziare le Consumed Thing esposte nel Thing Module, per gestirne l'interazione.

2. Configurare parametri logici per la gestione e il setting della comunicazione con una delle Thing Description note al sistema.

3. Raccogliere i dati di monitoraggio e formattarli in file di tracciamento.

4. Invocare i metodi di addestramento nel Learning Module, per definire i parametri secondo i criteri di ottimizzazione sui modelli forniti nella Thing Description.

5. Generazione di un Web Digital Twin a partire dal modello del Learning Module, con relativa interfaccia per l'interazione con servizi esterni, basata su una Thing Description definita in dipendenza della Web Thing.

6. Supporto scalabile verso la gestione di multipli WDT caricati all'interno del sistema.

7. Parsing di collezione di azioni e proprietá, in modo da rendere i dati chiari e ben formattati per la corretta fruizione dei servizi del sistema.

8. Inserimento e lettura dei modelli di ottimizzazione da e verso Redis.

In conclusione, possiamo dire che tale componente sia responsabile dei servizi critici del sistema, provvedendo alla raccolta o alla distribuzione dati.

Thing Module

Tale modulo é responsabile della generazione e l'esposizione delle operazioni di lettura, scrittura ed esecuzione di azioni definite nelle Thing Description. La Thing Module definisce l'interazione con le Web Thing connesse, attraverso l'utilizzo di Thing logiche esposte, in relazione alle invocazioni di azioni e operazioni di lettura e scrittura di proprietá del Core Module. Comunica con i dispositivi fisici per leggere i valori direttamente dai sensori oppure li gestisce logicamente attraverso un Simulation Module. La scelta della configurazione e della struttura interna al modulo é dipendente dal tipo di caso d'uso analizzato.

Simulation Module

Provvede a fornire un interfaccia di gestione dati capace di poter definire un comportamento simulato di un sensore o di un attuatore reale. Definisce una distribuzione dei valori che, a differenza dei dati letti dai sensori fisici, non sono vincolati dal tempo di campionamento, ma solamente ai valori di uno o piú stati precedenti. In altri termini, si basano su distribuzioni statistiche con parametri logici, in grado di generare valori e definire la loro derivazioni nel tempo. La formulazione della distribuzione dei valori di una componenti, per poter seguire una logica coerente con il reale comportamento dell'oggetto simulato, si basano sull'applicazione di leggi fisiche o matematiche.

Learning Module

Il Learning Module ha come obiettivo finale quello di definire il modello su cui si baserá il Digital Twin. Nella sezione 3.4 verranno introdotte delle estensioni sui metadati in grado di definire un modello capace di descrivere matematicamente il comportamento di una specifica proprietá, in relazione alla distribuzione dei suoi valori sulla Web Thing a cui si fa riferimento. Il Learning Module riceve un file .csv contenente dati monitorati o simulati in relazione al tipo di acquisizione e alla Thing Description di riferimento. Si ha la possibilitá di poter eseguire direttamente il modulo in locale su un documento che listi le osservazioni e le azioni di un dispositivo, anche in modalitá offline; avente, peró, il vincolo di avere comunque a disposizione la Thing Description di riferimento alla cronologia di azioni e di proprietá che si vuole analizzare. Il Learning Module raffigura una delle parti piú importanti dell'intero progetto. Le sue funzionalitá sono illustrate ampiamente all'interno della sezione 3.5.

Twin Module

Il Twin Module definisce un interfacciamento verso un tipo di una Web Digital Twin del sistema. Come per la Thing Module, anche il Twin Module sfrutta le WoT Scripting API per poter leggere e scrivere proprietá o invocare azioni sulle risorse a cui é associato. Tali risorse, nel caso del Twin Module, fanno riferimento al modello del Learning Module. In altri termini, i valori letti dalle varie proprietá e lo stato di invocazione di un azione, vengono definiti in relazione al modello comportamentale legato al WDT e non

da sensori o attuatori fisici e simulati. Esternamente, non vi sono distinzioni tra le interazioni tra Web Thing e Web Digital Twin, questo perché la comunicazione avviene attraverso simili WoT Scripting API che, peró, vengono implementate diversamente al loro interno. Per poter invocare un azione o una richiesta di lettura di una proprietá, bisogna definire alcune proprietá comuni ad ogni interfaccia del modulo:

1. **ID del modello di ottimizzazione:** Esso definisce quale modello di ottimizzazione dover richiamare per poter adottare una corretta associazione tra il WDT e il suo stato nel tempo.

2. **Istante Temporale:** Poiché i dati del modello sono orientati dal tempo, bisogna specificare l'instante temporale in cui si vuole effettuare una operazione.

Ogni modello viene generato, associato e salvato all'interno del Learning Module che, rispettando il pattern di comunicazione dell'architettura, é raggiungibile dal Twin Module solamente tramite l'invocazione di REST API, definite all'interno del Core Module. La chiamata a tali API viene implementata dagli handler definiti internamente alle WoT Scripting API. Tali funzioni possono venir sfruttati da piú Web Digital Twin di uno stesso tipo (ossia basati sulla descrizione di una medesima Thing Description), andando a classificare i vari WDT in base all'identificativo del modello nelle sue proprietá.

Redis Store

Uno dei nuovi moduli integrati nella nuova versione di Relativistic Twin é il Redis Store. Esso ha il compito di leggere e scrivere le informazioni dei modelli del Learning Module, in modo da poter mantenere in maniera persistente i WDT creati durante l'utilizzo del framework. Essendo un database chiave-valore, Redis associa ad ogni identificativo un valore contenente, in formato JSON, tutte le informazioni per ricaricare il modello come, ad esempio: Thing Description, dati osservati dal modello o i parametri di ottimizzazione. L'unico modulo che puó accedere direttamente al database é il Core Module. Le interazioni con gli altri moduli avviene solamente tramite API, questo permette di far gestire la comunicazione, e lo stato di persistenza dei modelli, ad un solo modulo del sistema, evitando criticitá e conflitti di scrittura e lettura.

Capitolo 4

Implementazione

4.1 Tecnologie Utilizzate

All'interno di questa sezione, descriveremo i framework e le maggiori librerie utilizzate per l'implementazione dei vari moduli dell'architettura. Nello specifico, andremo a definire le loro caratteristiche principali per poi descrivere come esse siano state utilizzate e concludere con la descrizione dell'implementazione dei singoli moduli sulle varie funzionalitá messe a disposizione.

4.1.1 AngularJS

AngularJS é un framework open-source mantenuto e aggiornato attualmente da Google. Esso é un framework strutturale per la creazione di applicazioni web dinamiche. Essendo stato implementato per definire pagine dinamicamente sul lato front-end, ha la funzione di estendere e migliorare alcune caratteristiche HTML. Rappresenta un alternativa sintetica e funzionale alla struttura statica e rigida di HTML, andandone a definire la logica, i dati e le componenti a runtime; basandosi su alcuni concetti fondamentali. Segue un modello MVC (Model-View-Controller) per la gestione delle componenti. Scritto in Javascript, riesce, tramite le sue caratteristiche principali, ad essere un framework adatto per l'implementazione di applicazioni web scalabili, ad alte prestazioni e con alta manutenibilitá.

Caratteristiche

Le sue caratteristiche di base sono:

1. **Data Binding:** In grado di definire in maniera sincrona ed automatica i dati da dover inserire in alcune componenti HTML seguendo logiche funzionali dei model e applicandoli alle view.

2. **Scope:** Le correlazioni tra controller e view vengono definiti in ambienti circoscritti allo scope della pagina visualizzata, rendendo il framework modulare.

3. **Controller:** Definiscono delle funzionalitá visibili ed applicabili solamente in un particolare scope logico dell'applicazione.

4. **Servizi:** AngularJS provvede l'utilizzo di funzioni built-in, generalmente seguendo il design pattern del Singleton, per poter definire in maniera semplice e riutilizzabile delle procedure comunemente utilizzate nelle applicazioni web come, ad esempio, l'invio di request ad un server back-end.

5. **Filtri:** Componenti in grado di filtrare elementi da una collezione basandosi su condizioni logiche.

6. **Direttive:** Definiscono tag, attributi o componenti HTML personalizzate in grado di poter essere riutilizzate all'interno del DOM di una applicazione web sviluppata.

7. **Template:** Raffigura una o piú pagine HTML in cui poter definire la composizione statica di una applicazione web.

8. **Deep Linking:** AngularJS é in grado di definire lo stato di una pagina visualizzata. Tramite URL é possibile variarne o recuperare uno stato specifico.

9. **Dependency Injection:** Si ha la possibilitá di definire un sottosistema di dependency-injection in grado di far creare, capire e testare un applicazione in maniera modulare e semplice.

Vantaggi

I vantaggi di AngularJS possono essere sintetizzati nei seguenti punti:

1. Creazione di pagine in maniera dinamica, semplice e mantenibile.
2. Capacitá di effettuare data binding sui contenuti e le caratteristiche dei tag HTML rendendo le pagine adattabile ai contesti di utilizzo.
3. Fornisce unit test per il debugging
4. Separazione dei concetti e gestione delle dipendenze tramite dependency injection
5. Componenti riutilizzabili (direttive) e strutturalmente configurabili (template)
6. Separazione dei controller in maniera granulare andando a definire la logica e la gestione di uno stato della pagina separatamente dalla view.

4.1.2 Node.js

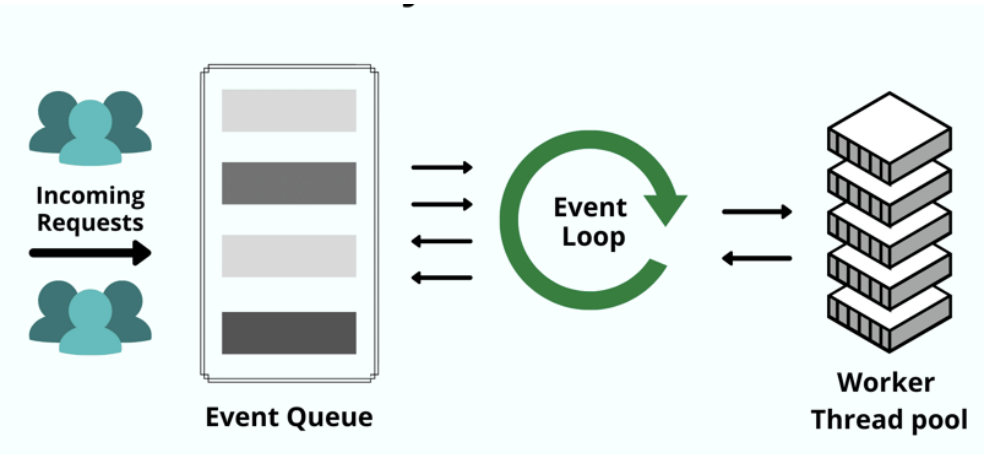

Figura 4.1: Architettura Single Threaded Event Loop di Node.js

Node.js é un framework che fornisce un ambiente single-threaded, open-source e multipiattaforma in grado di creare applicazioni server-side veloci e scalabili. Gira su un runtime Javascript e la sua architettura event-driven e non bloccante rende le applicazioni efficienti e adatte per contesti real-time.

Architettura

L'architettura utilizzata é chiamata Single Threaded Event Loop (Fig. 4.1) ed é in grado di gestire piú client parallelamente allo stesso tempo. Rispetto ad architetture multi-thread, in cui ad ogni richiesta concorrente al server si fa affidamento a piú thread eseguiti parallelamente, Node.js utilizza un procedimento differene basato sui seguenti passaggi:

1. Si mantiene una thread pool limitata
2. Ogni volta che arriva una richiesta client, essa viene inserita in una coda
3. Un processo chiamato "Event Loop" controlla ciclicamente se vi siano richieste in coda
4. Quando arriva una richiesta, verifica se essa sia bloccante o meno. Nel caso lo sia, viene assegnato un thread dalla thread pool per mantenere le caratteristiche non bloccanti del framework. I thread sono limitati, quindi un server puó ricevere e gestire simultaneamente solamene un numero finito di richieste bloccanti. Il gruppo di thread ausiliari utilizzati per la gestione delle richieste bloccanti vengono chiamati worker group.
5. Per garantire scalabilitá, é possibile estendere le risorse utilizzate in modo da supportare una quantitá complessiva di richieste bloccanti maggiore.

Tali caratteristiche hanno come vantaggio l'utilizzo limitato di risorse in grado di ottimizzare il tempo di esecuzione delle procedure legate alle richieste client, donando supportabilitá verso applicazioni web in cui l'adattabilitá real-time risulta essere critica. La scelta di Node.js é adeguata per il caso d'uso di nostro interesse dato che, considerando le procedure real-time per il monitoraggio di Web Thing e per l'interazione con le Web Digital Twin, risulta essere conforme con i requisiti funzionali del sistema prodotto.

Caratteristiche

Oggigiorno, Node.js é diventato uno dei framework server-side piú utilizzati. Le caratteristiche che lo contraddistinguono da framework concorrenti possono essere sintetizzate nei seguenti punti:

1. **Semplicitá:** Grazie alla documentazione e a fonti terze sempre piú approfondite ed aggiornate e data la semplicitá di implementazione, Node.js risulta essere semplice da imparare e da configurare, andando a definire applicazioni modulari e facili da comprendere.

2. **Scalabilitá:** Data l'architettura single-thread che garantisce un parallelismo per multiple richieste servite nello stesso istante, é capace di poter gestire un enorme numero di connessioni simultaneamente.

3. **Velocitá:** Data l'esecuzione non bloccante dei thread, ottimizza le procedure di business rendendole efficienti e veloci.

4. **Modularitá ed Estendibilitá:** Grazie al supporto verso l'ecosistema NPM[1], Node.js é in grado di utilizzare qualsiasi pacchetto al suo interno andando ad estendere le funzionalitá legate alle API che espone.

5. **Solido:** Anche se Node.js viene eseguito su un motore Javascript, la sua implementazione a basso livello é in C e C++, andando ad ottimizzare l'utilizzo della memoria, i tempi di esecuzione e di gestione degli eventi e delle risorse utilizzate e largamente descritte nella sua architettura.

6. **Supportabilitá:** Il supporto verso multipiattaforme garantisce l'applicabilitá e la supportabilitá per creare piattafome SaaS[2], applicazioni web o mobili.

7. **Mantenibilitá:** Poiché sia il frontend che il backend possono essere gestiti esclusivamente con Javascript, Node.js risulta essere semplice da utilizzare per gli sviluppatori web, rendendo le procedure di business frontend e backend facili da integrare.

4.1.3 Nest.js

Nest.js é un framework per costruire applicazioni server-side, basate su Node.js, che siano scalabili ed efficienti. Implementato in Javascript progressivo con un supporto completo verso Typescript, combina elementi del

[1]Gestore di pacchetti per il linguaggio di programmazione JavaScript; é il gestore di pacchetti predefinito per l'ambiente di runtime JavaScript Node.js ed é in grado di installare librerie e gestirne le dipendenze.

[2]Software as a Service, prevede l'utilizzo modulare di un applicazione fornita da terze parti in maniera riutilizzabile e adattabile.

paradigma orientato agli oggetti (PPO) con programmazione funzionale (PF) e programmazione reattiva funzionale (PRF). Nest.js é estendibile verso altre librerie grazie al suo ecosistema adattabile, con un supporto completo verso framework Javascript. Generalmente, viene utilizzato insieme a framework frontend come AngularJS, ReactJS o Vue ed é in grado di fornire dependency injection integrati per il coordinamento di servizi con tali tecnologie.

Caratteristiche

Le principali caratteristiche che contraddistinguono Nest.js da altri framework concorrenti possono essere sintetizzate nei seguenti punti:

1. Esso é un framework facile da utilizzare, intuitivo e semplice da padroneggiare

2. Ha una interfaccia a riga di comando (CLI) in grado di fornire funzionalitá aggiuntive in modo tale da facilitare il processo di sviluppo di un applicazione

3. Fornisce funzionalitá in grado di ottimizzare l'applicazione di unit test in maniera semplice e modulare.

4. Implementato su Typescript, andando a sfruttare la tipizzazione del linguaggio che la contraddistingue da Javascript.

5. Supporto verso moduli specifici di tecnologie emergenti come GraphQL, Mongoose, servizi di registrazione, convalida, memorizzazione della cache.

Una delle caratteristiche logiche che contraddistingue Nest.js da altri framework simili é data dalla struttura a componenti formata da 3 tipi principali:

1. **Controller:** Il sistema di routing di Nest.js seleziona un controller per ogni richiesta HTTP da parte di un client, il controller definisce la business logic di una richiesta per formulare poi una risposta e reinviarla al client richiedente.

2. **Provider:** Classe di un insieme di componenti che raffigurano gli elementi logici chiave di Nest.js. Essi sono generalmente servizi, repository, helper o altre componenti che possono essere iniettate come

dipendenza in oggetti andando a definire la loro intercomunicazione all'interno dell'ambiente runtime di Nest.js.

3. **Moduli:** Sono dei decorator che estendono delle proprietá comuni a delle componenti a cui vengono associate. Definiscono dei metadati a Nest.js per identificare quali componenti, controller o altre risorse verranno utilizzati nel codice dell'applicazione e combinarsi in un unico set.

4.1.4 node-wot

Sviluppato dalla Eclipse Foundation, il progetto node-wot raffigura l'implementazione ufficiale di riferimento del modello di iterazione del Web of Things descritto nel capitolo 2 ed esposto dalla W3C. Tale implementazione sviluppa tutte le caratteristiche di base del WoT andando ad implementare funzioni come:

1. Parsing e serializzazione di Thing Description
2. Supporto verso diversi Protocol Bindings per l'associazione con i protocolli piú diffusi nelle applicazioni IoT moderne
3. Implementazione del sistema di runtime (WoT Runtime) che fornisce le Scripting API per applicare le funzionalitá relative a Thing esposte o consumate.
4. Toolkit di visualizzazione delle informazioni di una Thing Description attraverso un interfaccia web browser chiamata WebUI.

4.1.5 Docker e Docker Swarm

All'interno di questa sezione introdurremo, brevemente, i concetti che identificano Docker e Docker Swarm, prendendo in considerazioni le caratteristiche principali con cui trarre beneficio nel loro utilizzo.

Docker

Docker rappresenta una piattaforma software in grado di creare, utilizzare e testare applicazioni all'interno di un ambiente controllato distribuibile tramite immagini. Le immagini rappresentano le istruzioni per costruire un

applicazione, andandone a rispettare le caratteristiche dell'ambiente e le dipendenze, in modo veloce e trasparente. Con Docker viene introdotto il termine di container. Un container non é altro che un ambiente protetto e isolato in grado di eseguire delle applicazioni. I container permettono anche

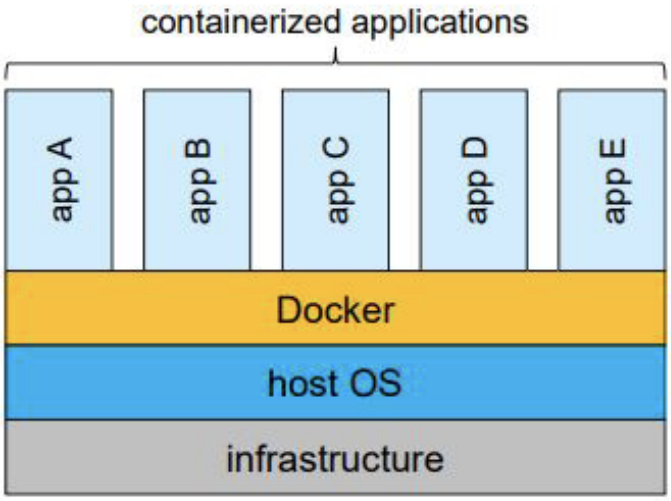

Figura 4.2: Stack Architetturale dei container Docker su una macchina virtuale

di distribuire il codice in maniera standarizzata andando ad ottimizare l'utilizzo delle risorse e rendere un applicazione utilizzabile in qualsiasi ambiente con particolare attenzione all'utilizzo di risorse; i container non sono onerosi in termini di consumo di risorse del sistema, andando a condividere solamente il kernel della macchina virtuale in cui vengono eseguiti. In figura 4.2 é possibile analizzare lo stack architetturale su cui si basa il processo di containerizzazione. Nello specifico possiamo sintetizzare le sue caratteristiche nei seguenti punti:

1. **Rapida Distribuzione di Software:** Tramite un immagine che definisce le caratteristiche dell'ambiente e delle dipendenze di un applcazione, é possibile costruire un container Docker in maniera trasparente dal sistema che esegue tale processo in maniera veloce ed efficiente.

2. **Funzionamento Standarizzato:** L'uso di container per le piccole applicazioni riesce a semplificare non solo la distribuzione ma anche il rilevamento di problemi con relativo rollback di ripristino.

3. **Trasferimento ottimizzato:** Le applicazioni su Docker possono migrare tra un nodo ad un altro molto facilmente, che esse siano su una macchina locale o distribuite su cloud.

4. **Riduzione dei costi:** I container Docker sempificano l'esecuzione di codice sui server, andandone a migliorare i livelli di utlizzo e contribuiscono alla gestione ottimale delle risorse.

I container Docker vengono utilizzati principalmente per la diffusione e la distribuzione di microservizi efficienti in grado di creare e gestire indipendentemente i moduli di un architettura orientata a servizi. Inoltre, essi possono fornire moduli standalone in grado di poter interfacciare un applicazione esternamente cosi da poterla utilizzare funzionalmente, andando a sfruttare i container distribuiti da terze parti.

Docker Swarm

Docker Swarm é uno strumento di gestione per container Docker chiamato orchestratore. Viene generalmente utilizzato per la gestione di container multipli che possono essere localizzati anche su differenti host della rete. Docker Swarm offre una gestione scalabile, in grado di poter definire un alta disponibilitá delle funzioni di applicazioni in un cluster, andando a definire un architettura che, ad alto livello, é rappresentata dall'intercommunicazione tra un nodo master e dei nodi workers; ossia si ha la possibilitá di far gestire ad una unitá centrale tutte le operazioni del cluster in maniera veloce, efficiente e trasparente. In Figura 4.3 é rappresentata l'architettura ad alto livello di Docker Swarm; possiamo notare come é possibile definire piú di un nodo master in grado di convidere uno stato globale cosi da poter scalare la gestione dei worker su piú nodi.

I servizi offerti da Docker Swarm sono principalmente applicazioni che vengono eseguite su nodi workers, essi possono essere di due tipi principali:

1. **Servizi Replicati:** In cui si ha una replica dei servizi e delle loro funzionalitá, in modo che una medesima funzione possa essere eseguita su piú worker, quando tale funzione verrá richiamata, il nodo manager assegnerá la richiesta di esecuzione al primo nodo worker disponibile che espone il servizio richiesto.

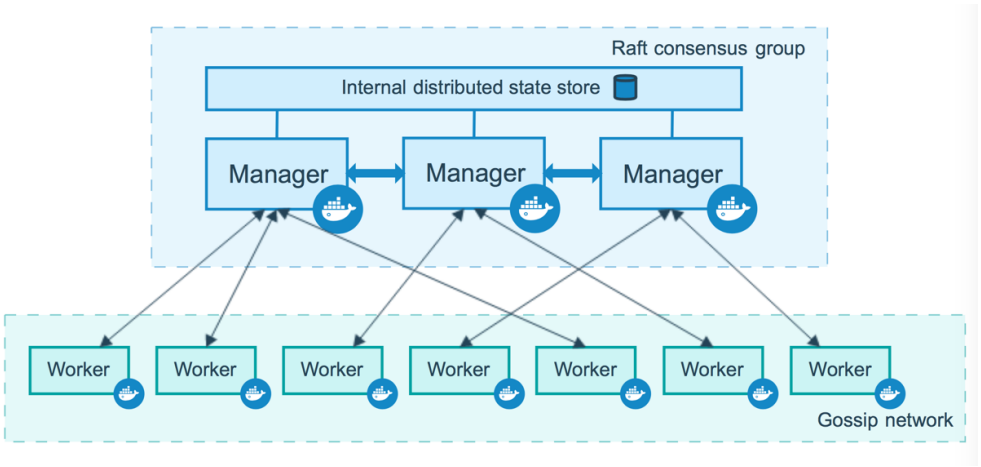

Figura 4.3: Architettura ad Alto Livello di Docker Swarm

2. **Servizi Globali:** Un servizio che é disponibile su tutti i nodi che soddisfano i requisiti non funzionali e vincoli relativi all'utilizzo di risorse.

Docker Swarm é in grado di poter gestire macchine virtuali o fisiche nello stesso ambiente, senza nessuna distinzione tra i tipi, andandone ad astrarre anche le caratteristiche hardware e software dell'ambiente, mantendendo solamente la gestione delle risorse in modo da poter rispettare i vincoli non funzionali delle applicazioni eseguite nei container all'interno dei vari worker.

4.1.6 Redis

Redis é un sistema di archiviazione di strutture di dati on-memory completamente open-source, utilizzato come:

1. **Database:** Mantenendo la persistenza dei dati secondo un approccio in-memory, spostando successivamente i dati nel disco per la sua persistenza.

2. **Cache:** Sfruttando la memoria del terminale in cui é installato, Redis puó mantenere dei dati temporaneamente, il tempo di mantenimento dei dati in cache é scelto a livello programmativo.

3. **Broker:** Puó essere utilizzato all'interno di un canale di comunicazione PubSub come broker per la comunicazione. Fornisce registrazioni a

canali e meccanismi di propagazione di messaggi in un ambiente event-driven.

Redis fornisce strutture di dati come stringhe, hash, liste, insiemi o insiemi ordinati, con query di intervallo, bitmap, hyperloglogs, indici geospaziali e flussi. É possibile eseguire operazioni atomiche su questi tipi, come l'aggiunta di una stringa, l'incremento del valore in un hash, l'aggiunta di un elemento a un elenco, il calcolo dell'intersezione, dell'unione e della differenza di un insieme o l'ottenimento di un membro con il ranking piú alto, in un insieme ordinato. Per ottenere le massime prestazioni, Redis lavora con un set di dati in memoria. A seconda del caso d'uso, Redis puó conservare i dati scaricando periodicamente il dataset su disco o aggiungendo ogni comando a un registro basato sul layer di persistenza del calcolatore. É anche possibile disabilitare il servizio di persistenza se si ha bisogno solo di una cache, ricca di funzionalitá e distribuibile nella rete. Redis supporta la replica asincrona, con una sincronizzazione veloce e non bloccante e la riconnessione automatica con risincronizzazione parziale, in caso di divisione della rete. All'interno dell'architettura proposta, utilizzeremo Redis per la persistenza dei modelli del WDT generati all'interno del Learning Module.

4.2 Implementazione dei Moduli

Questa sezione illustrerá come siano applicate le tecnologie definite nella sezione 4.1; andando a descrivere l'implementazione dei vari moduli, lo sviluppo dei servizi e le caratteristiche di ogni funzionalitá definita nell'architettura proposta in sezione 3.

4.2.1 Dashboard Module

L'implementazione della dashboard é completamente in Angular. Come descritto nella sezione 4.1.1, la divisione degli elementi dell'interfaccia é divisa in componenti logiche, in cui la gestione é, generalmente, definita all'interno di servizi, che assumono un ruolo specifico per la gestione dell'interazione con l'utente finale. Come descritto in sezione 3.7.1, andremo a descrivere due differenti tipi di dashboard e dividere le funzionalitá di gestione delle Web Thing dalle Digital Web Thing (WDT).

Dashboard: Web Thing

L'interfaccia definisce varie funzionalitá:

1. Scegliere la Web Thing, in relazione al tipo di Thing Description selezionata.
2. Visualizzare le proprietá della TD, insieme a metadati quali: nome, descrizione e tipo.
3. Definire e gestire i form per l'esecuzione di azioni all'interno dell'interfaccia
4. Avviare una fase di monitoraggio, attraverso un form in cui specificare: tempo iniziale, tempo finale e frequenza di campionamento, in relazione ad una sessione di monitoraggio.
5. Caricare e scaricare dati di campionamento da e verso la sessione di monitoraggio corrente.
6. Addestrare un modello nel Learning Module a partire dai dati di monitoraggio della sessione.

Tali funzionalitá verranno descritti nei successivi paragrafi in relazione alle decisioni implementative.

Selezione Thing Description Una componente $MonitorFormComponent$ provvede a fornire un form in grado di selezionare il titolo da una collezione di Thing Description, caricati allo startup come variabili di istanza, per andare a selezionare i metadati da dover visualizzare all'interno della pagina. Il form rimane visibile anche dopo il selezionamento, questo permette di poter cambiare il tipo di Thing Description, e della Web Thing annessa, in qualsiasi momento. I dati delle Thing Description vengono salvati sul *MonitorFormComponent*, in una lista di oggetti $ThingDescription$, secondo il formato JSON fornito dal *node-wot*.

```
1 const td : ThingDescription = {
2     title: TITLE,
3     description: DESCRIPTION,
4     // ...
5     properties : {
6         NAME-PROPERTY : {
```

```
7         type: TYPE-PROPERTY,
8         readOnly: true|false,
9         writeOnly: true|false,
10        // ...
11    }},
12    actions : [{
13       ACTION-NAME : {
14            description: ACTION-DESCRIPTION,
15            input : {
16                type: INPUT-TYPE
17            }
18       }
19    }]
20    // ...
21
22 }
```

Listing 4.1: Struttura Generale di una Thing Descriptions

La selezione del TD avviene all'interno del metodo $switchTD()$ della classe $MonitorFormComponent$.

Visualizzazione delle proprietá di un TD Una volta selezionata la Thing Description, si ha la possibilitá di visualizzare in formato tabellare le sue proprietá. La tabella mostra tre tipi di informazione: il nome della proprietá, la descrizione e il tipo. I dati vengono prelevati dall'oggetto $ThingDescription$ selezionato.

Gestione e invocazione di azioni Le azioni definiscono un altro importante elemento da dover visualizzare. Per poter interagire con la Web Thing e dare la possibilitá di eseguire delle operazioni, sono stati implementati dei form a partire da una collezione $FormField$ che definisce i metadati utili per la fruizione di tale servizio.

```
1 export class FormField {
2    fieldName : string;
3    fieldDescription : string;
4    input : InputField; // {name, type}
5 }
```

Listing 4.2: Classe FormField

Ad ogni invocazione di un azione, si richiama un servizio $SettingTDService$ che, tramite un servizio $NetworkService$, effettua una request HTTP per

l'esecuzione dell'azione sulla Web Thing, definita in backend. Il tempo di invocazione dell'azione é strettamente correlato al tempo corrente della Web Thing, gestibile attraverso una specifica azione dell'interfaccia.

Monitoraggio Per poter iniziare o continuare una fase di campionamento sulle proprietá della Web Thing, l'interfaccia dispone di un form di gestione per tale servizio. Essa é formata da tre campi principali:

1. **Tempo di inizio:** Che definisce l'istante che bisogna attendere per poter iniziare a registrare i dati dalla lettura delle proprietá
2. **Tempo di fine:** Che identifica il tempo massimo da raggiungere prima di concludere una sessione di monitoraggio
3. **Frequenza:** In grado di specificare il numero di millisecondi che bisogna far passare tra due letture consecutive.

Il form gestisce il tempo in millisecondi. Una volta che la fase di monitoraggio é iniziata, i dati vengono salvati in un oggetto dentro la sessione del Core Module. L'intero processo é gestito all'interno del metodo $submitProcedure()$ del $MonitorFormComponent$.

Gestione Dati di Campionamento Una volta terminata una sessione, i dati possono essere salvati in un file in formato csv, per poi poter essere scaricati sul terminale dell'utente. Questo permetterá di poter utilizzare i dati sia per servizi interni, che esterni al framework. La formattazione dei dati e la creazione del file csv vengono effettuati da alcuni metodi di $MonitorFormComponent$:

1. $formatHistory()$ si occupa della formattazione dei dati di campionamento in un formato JSON supportabile dalla procedura di conversione in csv implementata.
2. $generateBlob()$ definisce il formato csv dei dati e li racchiude all'interno di un oggetto $Blob$.
3. $retrievesHistory()$ si occupa di definire un URL temporaneo che, automaticamente, avvia il download del file csv con i dati formattati del $Blob$.

```
1
2   async retrievesHistory() {
3     this.settingTDService.retrievesHistory().then((history: any) => {
4       history = this.formatHistory(history);
5       const filename = 'history.csv'
6       var blob = this.generateBlob(history, true)
7       const link = document.createElement('a');
8       if (link.download !== undefined) {
9         const url = URL.createObjectURL(blob);
10        link.setAttribute('href', url);
11        link.setAttribute('download', filename);
12        link.style.visibility = 'hidden';
13        document.body.appendChild(link);
14        link.click();
15        document.body.removeChild(link);
16      }
17      return true;
18    })
19  }
```

Listing 4.3: Metodo per scaricare la sessione di campionamento in un file CSV

Oltre alla funzione di download dei dati, si ha la possibilitá di caricare una sessione a partire da un file CSV con la stessa struttura logica supportata dal sistema. In altri termini, si ha la possibilitá di ricaricare una sessione di monitoraggio precedente, in modo da poter recuperare campionamenti antecedenti al tempo corrente. Il metodo che si occupa di tale funzionalitá é *loadMonitorData()*, definito all'interno della classe *MonitorFormComponent*. Una volta caricati, i dati verranno inviati al backend per poter essere inseriti alla lista di osservazioni legate alla sessione di monitoraggio.

Addestramento di un Modello Una volta completata la fase di monitoraggio, l'utente puó richiedere l'addestramento di un modello. A partire dai dati di campionamento nella sessione, il learning Module andrá a definire training e testing set. L'input di tale azione viene definito da un semplice bottone. Al suo click, si invoca la funzione *learningFromData()*, di *MonitorFormComponent*, che richiama un servizio di *settingTDService* per la gestione dei dati della richiesta, e *NetworkService* per la richiesta di training via HTTP verso il Core Module.
Ogni form presente all'interno della dashboard é gestito da un oggetto *For-*

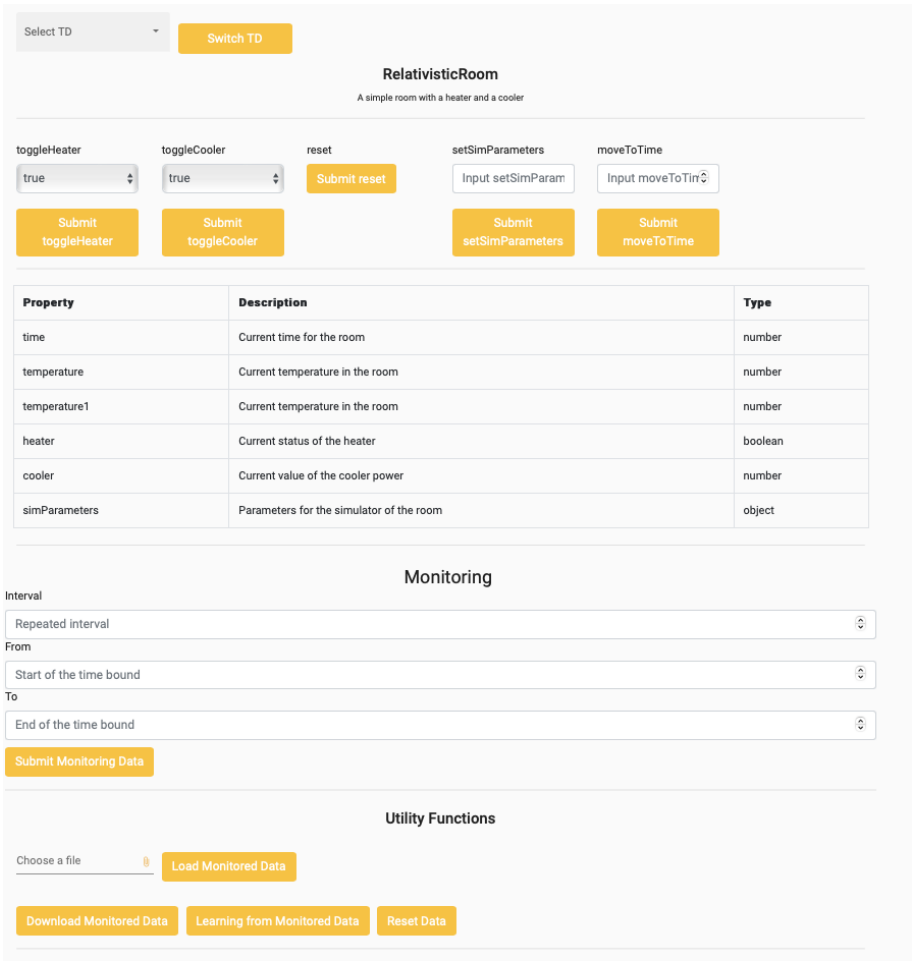

Figura 4.4: Interfaccia di Monitoraggio per la Web Thing definita dal TD RelativisticRoom

mGroup, i vari input vengono definiti attraverso la classe *FormControl*. I dati e lo stato di ogni campo, vengono manipolati all'interno dei metodi di *MonitorFormControl*. Infine, un servizio *EventBusService* gestisce la visualizzazione dello stato dell'interfaccia e dell'esito delle operazioni effettuate.

4.2.2 Dashboard: Web Digital Twin

Se il sistema dispone di almeno un Web Digital Twin, addestrato precedentemente sui dati della sessione di campionamento di una delle Web Thing

del sistema, si ha la possibilitá di utilizzare una interfaccia dedicata ai WDT. Essa dispone di alcune funzionalitá, giá introdotte in sezione 3.7.1:

1. Selezionare il tipo di WDT attraverso la scelta delle Thing Description con almeno un twin disponibile nel sistema.

2. Eseguire, cancellare e visualizzare azioni su un WDT selezionato

3. Definire il grafico di predizione, in relazione ai dati reali precedentemente monitorati sulla WT.

4. Visualizzare, in versione semplice o estesa, il diagramma di variazione di stato delle WDT selezionate.

5. Definire thresholds per analizzare l'andamento degli stati in relazione a vincoli inferiori e/o superiori.

I servizi di visualizzazione delle opzioni per la gestione del grafico di predizione o del diagramma di stato legato ai WDT, viene effettuato tramite una selection all'interno dell'interfaccia.

Selezione del Tipo di WDT In maniera simile alla selezione della Web Thing per l'interfaccia di monitoraggio 4.2.1, si utilizza un form per selezionare le Thing Description dei WDT caricati come oggetti $ThingDescription$ (4.1) all'interno del $TwinComponent$ in fase di startup. Una volta selezionato uno tra i servizi (Gestione dei diagrammi di predizione o dei diagrammi di stato), verranno visualizzate tutte le componenti secondo i metadati annessi alla TD selezionata. La selezione dei servizi e del tipo di WDT vengono gestiti dal metodo *changeView()* del *TwinComponent*.

Gestione delle azioni Una volta definito il tipo di TD per i WDT dell'interfaccia, verranno visualizzate le seguenti componenti:

1. Una lista di form gestiti da oggetti $FormGroup$ e $FormControl$ nella componente $TwinComponent$, in relazione alle informazioni del campo *actions* dell'oggetto *ThingDescription* selezionato. Ogni form ha un campo *select* per selezionare uno specifico WDT in cui compiere l'azione data in input.

2. Una tabella con la lista delle azioni, dei dati di input e del timestamp di esecuzione. La tabella fa riferimento ad una WDT opportunatamene selezionata da un campo *select*. Inoltre, ogni riga ha un bottone associato per la cancellazione dell'azione della stessa riga.

Tramite tali componenti, si ha la possibilitá di gestire completamente le azioni collegate ad un Web Digital Twin e, a secondo del tipo di servizio selezionato, si puó visualizzare il risultato di un azione sull'andamento delle varie proprietá, o sul cambiamento di stato della WDT. Dato il nominativo dell'action sulla TD e i dati in input, l'invocazione di un azione viene effettuata dal metodo $invokeAction()$ di $TwinComponent$. La cancellazione delle azioni avviene tramite un invocazione di $onDelete()$ con l'indice della riga da cancellare. In seguito ad un inserimento o una cancellazione, attraverso il servizio $TwinService$, si invia una richiesta di aggiornamento dello stato del modello, localizzato nel Learning Module.

Grafico di Predizione Selezionando il servizio di predizione, si ha la possibilitá di visualizzare un grafico su un intervallo di tempo dato in input dall'utente. Per poter visualizzare tale grafico, bisogna compilare un form con i campi per definire l'intervallo temporale di predizione, i vari WDT da utilizzare, un campo facoltativo di inserimento di un file con la lista di azione da dover eseguire. Inoltre, la visualizzazione puó unire i valori dei vari WDT in un unico grafico o in grafici consecutivi. Inoltre, un campo *select* definisce la selezione per tali opzioni. I valori andranno in input ad un metodo $getData()$ di $TwinComponent$, in cui si preleveranno, tramite i vari $formControl$, i dati inseriti e, usando un metodo di $TwinService$, si acquisiscono i dati di predizione dal Core Module. In seguito, sfruttando *Apache ECharts*, si formattano i dati per poterli visualizzare graficamente.

Diagramma di stato Selezionando il servizio per la generazione del diagramma di stato, si ha la possibilitá di visualizzare un grafico contenente gli stati dei WDT selezionati, in relazione ad una frequenza ed un tempo limite dato in input. Lo stato sará rappresentato da una serie di proprietá selezionabili in un campo *select*. Le proprietá vengono definite a partire dal tipo di Thing Description affiliato al WDT scelto. Inoltre, come per il grafico delle predizioni, si ha la possibilitá di aggiungere una lista di azioni da file e visualizzare il grafico in due differenti modalitá:

1. **Compacted:** In grado di definire i nodi in relazione alla variazione dello stato, rispetto ai vari vincoli inferiori e superiori definiti come threshold.

2. **Normal:** Definisce un grafico in cui ogni nodo rappresenta uno snapshot dello stato del WDT ad un determinato istante di tempo secondo i valori di frequenza e di tempo limite in input.

La gestione del diagramma di stato viene definita internamente al metodo *getData*() di *TwinComponent*. All'interno di tale metodo, si invocano funzionalitá del Core Module, per poter prelevare le informazioni legate ad un intervallo di tempo definito dai parametri: tempo di frequenza e tempo limite. Successivamente, i dati andranno ad essere gestiti dal metodo *generateStatesOptions*() di un servizio *StateService* che, in dipendenza della modalitá compacted o normal, genererá la lista di nodi e archi del diagramma di stato, secondo le regole descritte nei punti 1. e 2. precedenti.

```
{ type: TYPE-PLOT,
  title: {
    text: TITLE-TEXT
  },
  legend: {
    data: LEGEND-OF-DATA
  },
  grid: { MARGIN-DATA },
  series: [DATA-COLLECTION]
}
```

Listing 4.4: Opzioni generiche per ogni tipo di grafico definito da Apache EChart

Lo snippet 4.4 mostra la struttura di base delle opzioni di un grafico generato con *Apache EChart*, il campo *series* raffigura i dati del grafico. Nel caso del diagramma di predizione, i dati sono raffigurati come una serie di collezioni di liste $[x, y]$ con x il tempo e y il valore per ogni proprietá associata ad un label nel campo *legend*. Nel caso del diagramma di stato, i dati sono raffigurati da una lista di nodi (contenenti un valore, un nome e la posizione $[x, y]$ nel grafico, e da una lista di archi (contenenti un label, un nodo di inizio e un nodo di fine, con riferimenti ai nomi dei nodi).

Gestione di Threshold Quando si analizza un diagramma di stato, generalmente, ci si interessa se un particolare nodo del WDT superi o meno

dei vincoli sui valori di alcune delle sue proprietá. Per soddisfare tale esigenza, abbiamo implementato un meccanismo di inserimento di vincoli superiori e inferiori su ogni proprietá leggibile e numerabile di un WDT. Tramite la funzione *initThreshold*(), invocata in fase di startup della componente *TwinComponent*, andiamo a definire, per ogni WDT, una soglia minima *min* e massima *max* infinitesimale. In altri termini, per ogni proprietá P_i di un WDT T_j, con i il numero di proprietá del TD del WDT e j il numero di WDT di uno stesso tipo, i threshold iniziali sono tutti uguali a $min_i = -\infty$ e $max_i = +\infty$. Selezionando un WDT dalle opzioni di un campo *select*, si passa alla visualizzazione di una tabella con: proprietá, i valori di minimo e i massimo e un form per la variazione di tali dati. I valori di minimo e massimo vengono gestiti da un metodo *changeThreshold*() che, a secondo dei valori letti da un oggetto *FormGroup* dedicato, modifica le soglie minime e massime delle proprietá del WDT. I threshold vengono dati in input al metodo *generateStatesOptions*() di *StateService* per la colorazione degli stati.

```
1  setColorNode(data: any, threshold: any) {
2     let properties : string[] = Object.keys(threshold)
3     let color : string = 'blue'
4     for (let prop of properties) {
5       if (data[prop] != undefined) {
6         let min = threshold[prop].min
7         let max = threshold[prop].max
8         if (data[prop] < min) {
9           if(color == 'red') return 'black'
10          else color = 'purple' // min underflow
11        } else if (data[prop] > max) {
12          if(color == 'purple') return 'black'
13          else color = 'red' // max overflow
14        }
15      }
16    }
17    if(color != 'red' && color != 'purple') return 'blue' // normal
18    else return color // red or purple
19  }
```

Listing 4.5: Metodo di StateService per l'assegnazione dei colori sui dati in relazione ai valori dei threshold

Le regole di colorazione sono definite nella sottosezione di 3.7.1. Lo snippet 4.5 raffigura l'implementazione di tali regole. Infine, ogni form definito all'in-

terno della dashboard del WDT, viene gestito sempre tramite *FormGroup*, mentre i suoi campi di input sono definiti da elementi *FormControl*, impostati dentro il *TwinComponent* in fase di startup.

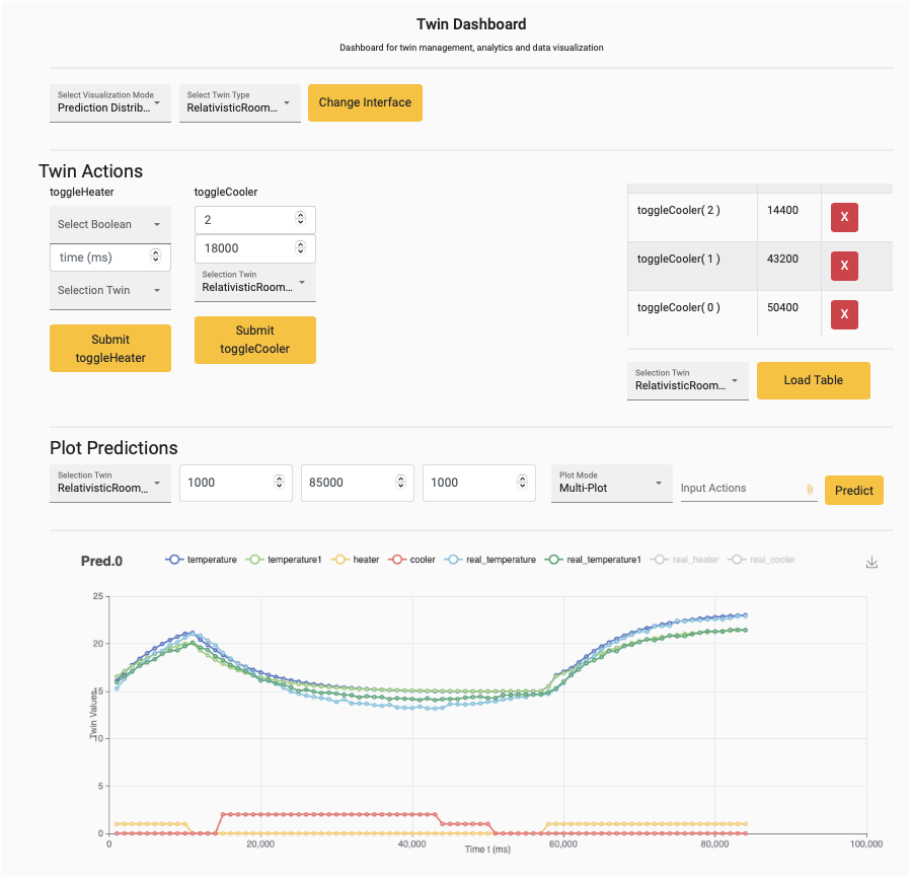

Figura 4.5: Snapshot della Dashboard per la gestione di WDT multipli

4.2.3 Comunicazione Tra Dashboard e Core Module

Ogni interazione tra la dashboard e il backend é definita attraverso una comunicazione HTTP. Per poter effettuare tale operazione, si sono definiti dei servizi per tutte le componenti Angular:

1. **Servizi di Gestione del Payload:** Nel caso si effettui una POST, i servizi di tale tipo (come *SettingTDService* o *TwinService*), hanno il compito di formattare i dati in modo da soddisfare i requisiti strutturali del payload della request.

2. **Servizio di Rete:** Esso é rappresentato da un solo servizio comune a tutte le componenti, *NetworkService*. Si occupa di inviare, tramite un oggetto *HttpClient*, delle richieste GET e POST a secondo del tipo di funzione invocabile da NestJS.

Tutte le funzionalitá, che usano una comunicazione su tali servizi, sono di tipo asincrono.

4.2.4 Thing Module

L'implementazione del Thing Module é in Typescript. Sfruttando il framework *node-wot*, si sono implementate degli oggetti $ExposedThing$ con $handlers$ per la lettura o scrittura delle proprietá ed esecuzione di azioni, che vanno a definire il comportamento della Web Thing in relazione alle operazioni espresse in un oggetto $WoT.ThingDescription$ definito secondo la struttura dello snippet 4.6

```
1  const td = {
2      title: "TITLE-TD",
3      description: "DESCRIPTION-TD",
4      properties: {
5          PROP-1 : {
6              type: "TYPE-PROP",
7              description: "DESCRIPTION-PROP",
8              observable: true|false,
9              readOnly: true|false,
10             writeOnly: true|false
11             uriVariables : {
12                 INPUT-1 : {type : "TYPE-INPUT-1"}
13             }
14         }
15         //...
16     },
17     actions: {
18         "ACTION-NAME": {
19             description: "ACTION-DESCRIPTION",
20             input: {
21                 type: "TYPE-INPUT-ACTION",
22             }
23         },
24     //...
25     }
```

```
26 }
```

Listing 4.6: Struttura di una WoT.ThingDescription per la generazione di un ExposedThing

Tramite l'interfaccia WoT, si ha la possibilitá di generare, a partire dalla Thing Description, un $ExposedThing$ su cui impostare gli handlers descritti precedentemente e, tramite la funzione $expose()$ renderla visibile sulla rete. Per il suo utilizzo, verranno invocati i metodi $readProperty()$ e $invokeAction()$ su un oggetto $ConsumedThing$, generato a partire dal medesimo TD, all'interno del Core Module. La gestione degli handlers dell'$ExposedThing$ é lasciata al programmatore che, rispettando le regole della Thing Description, potrá definire le procedure per il calcolo dei valori legati alle proprietá.

```
1       thing : ExposedThing;
2       //...
3       thing.setPropertyReadHandler("PROP-NAME", async (options) => {
4           return VALUE-PROP-NAME;
5       });
6       //...
7
8       this.thing.setActionHandler("ACTION-NAME", async (value, options)
            => {
9           const input = value.input
10          //... do some operations with input
11          return true
12      });
```

Listing 4.7: Aggiunta di un handler per la lettura di una proprietá e l'invocazione di un azione generica.

Tramite le proprietá di una TD, si possono definire anche dei valori utili per la gestione logica interna alla Web Thing. Un esempio, é fornire, tramite proprietá, le informazioni di inizializzazione di un simulatore, cosi da poter determinarne il comportamento alla variazione dei parametri matematici forniti.

4.2.5 Learning Module

Il Learning Module é definito da due differenti componenti:

1. Un server Flask per l'interfacciamento verso il Core Module

2. Una classe Optimizer per la generazione, gestione e mantenimento di un modello.

Server Flask

Il server flask mantiene, all'interno della sessione, dei modelli generati durante l'utilizzo del framework o caricati da un database Redis. Ogni modello é associato ad un WDT del Core Module, andandone a specificare il comportamento in base all'inferenza dei valori delle proprietá e all'invocazione di azioni, con parametri ottimizzati. In altri termini, il server espone tutte quelle funzionalitá per andare ad eseguire le operazioni del Web Digital Twin (invocazione di azioni o lettura di proprietá) sul modello comportamentale associato. All'interno del server Flask sono presenti vari API, invocabili tramite i seguenti percorsi:

1. */*: Per la generazione di un modello e la visualizzazione di un grafico di predizione e un analisi sulle performance dell'accuratezza dei modelli forniti nei campi "model" della Thing Description, modificata secondo i criteri espressi in sezione 3.4. Viene anche utilizzato dal Simulation Module per la generazione di grafici sull'andamento dei valori di un WDT temporaneo.

2. */learn*: Per l'inizializzazione e il salvataggio di un modello all'interno della sessione e all'interno del database Redis.

3. */invokeAction*: Per l'invocazione di un azione su un modello, dato il suo id in input.

4. */readProperty*: Per l'invocazione di un'operazione di lettura su un modello con id, tempo e nome della proprietá dato in input.

5. */getObservationData*: Per poter prelevare i dati di training su cui é stato addestrato il modello.

6. */addActionsList*: Per l'esecuzione di una lista di azioni, a determinati intervalli di tempo, su un modello con id specificato in input. Le azioni vengono inviate all'interno di una lista in un oggetto JSON nel payload della richiesta.

7. */deleteAction*: Dato l'id del modello e l'indice di un azione, si cancella l'elemento corrispondente all'indice in input all'interno della lista di

azioni dell'istanza avente l'id uguale a quello definito all'interno della richiesta.

8. */getDifProperties*: Dato un id di un modello, restituisce la lista dei nomi delle proprietá differenziali secondo la definizione descritta nella sezione 3.3.

9. */getActions*: Dato l'id di un modello in input, restituisce la lista di azioni correttamente registrate sul modello.

10. */getAlgProperties*: Dato un id di un modello, restituisce la lista dei nomi delle proprietá algebriche secondo la definizione descritta nella sezione 3.3

11. */getCurrentModelsNumber*: Richiesta che restituisce il numero dei modelli attualmente mantenuti nel Learning Module. Identifica, in maniera implicita, anche il numero di WDT presenti all'interno del sistema.

Allo startup dei moduli, il server procede alla lettura dei modelli sul database Redis, per poi renderli raggiungibili dai servizi del frontend. La lettura e la scrittura di modelli tra Redis e il server del Learning Module é gestita dalle componenti *ModelTwinService* e *ThingController* del Core Module.

Optimizer

La classe Optimizer é definita in Python. Essa é viene utilizzata per instanziare i modelli e fornisce un insieme di metodi per la gestione delle operazioni, in linea con le interazioni dei WDT. Nello specifico, la classe contiene i seguenti metodi:

1. *getThingDescription()*: Ritorna il titolo della Thing Description su cui si basa il modello. Essa combacia con la Thing Description modificata a partire dalla TD delle WT che forniscono i dati di addestramento al modello.

2. *getObservationData()*: Restituisce l'insieme dei dati osservati che raffigurano il training e il testing set del modello.

3. *getDifProperties()*: Restituisce l'insieme delle proprietá differenziali secondo le specifiche dei modelli forniti all'interno del TD esteso.

4. *getAlgProperties()*: Restituisce l'insieme delle proprietá algebriche secondo le specifiche dei modelli forniti all'interno del TD.

5. *getActions()*: Restituisce tutte le azioni registrate correttamente sul modello per la variazione dei valori predetti, in dipendenza dello stato delle proprietá algebriche.

6. *deleteAction()*: Metodo che, dato l'indice di un elemento all'interno della lista delle azioni del modello, elimina tale valore.

7. *getCurrentActions()*: Restituisce un sottoinsieme di azioni eseguite fino ad un timestamp dato in input.

8. *Fmodel()*: Esegue una equazione differenziale correttamente tradotta dal modello della Thing Description estesa, restituisce l'insieme delle proprietá differenziali secondo le specifiche dei modelli forniti all'interno del TD esteso.

9. *algFModel()*: Esegue una equazione algebrica correttamente tradotta dal modello della Thing Description estesa.

10. *integrateEstimationModel()*: Dati in input i valori P e $R(t_0)$, definiti secondo i criteri epressi in 3.5.2, e un dominio P superiormente e inferiormente limitato, risolve un sistema di ODE (equazioni differenziali ordinarie) sui parametri iniziali, date dalla soluzione della funzione *scipy.integrate.solve_ivp*.

11. *odefun()*: Definisce una equazione differenziale ordinaria appartenente al sistema di ODE dato a $integrateEstimationModel()$.

12. *extractParams()*: Estrae i valori dei parametri dei modelli della TD estesa, definiti secondo i criteri descritti in sezione 3.4, del modello di ottimizzazione.

13. *lsfun()*: Computa, a partire dai valori osservati e dai parametri dei modelli dell'estensione della WoT TD, la funzione per l'esecuzione dell'algoritmo di least squares non lineare.

14. *resetGlobal()*: Reinizializza il modello di ottimizzazione.

15. *splittingData()*: Dato un insieme di dati di osservazione, divide l'insieme in training set e testing set.

16. *init()*: Inizializza un modello addestrato. Andando a definire i parametri di ottimizzazione $\hat{P}$ e $\hat{R}(t_0)$ definiti secondo la procedura descritta in 3.5.

17. *main()*: Inizializza un modello, allo stesso modo di *init()*, inserendo i risultati in un payload per la visualizzazione su uno *scatterplot*.

18. *addActionsList()*: Aggiunge una lista di azioni alla collezione corrente del modello.

Le funzioni descritte sono interfacciabili esternamente all'interno di una REST API sul server Flask, in maniera diretta (tramite invocazione) o indiretta (tramite l'utilizzo di un metodo che ne invoca una istanza al suo interno).

4.2.6 Redis Store

Per poter salvare lo stato dei modelli del Learning Module, si fa affidamento ad una istanza di Redis. Essa é mantenuta all'interno di un container Docker, visibile all'interno del sistema. L'unico modulo in grado di interfacciarsi direttamente con il database é il Core Module che, tramite un servizio *ModelTwinService*, esegue le seguenti operazioni:

1. Scrittura o aggiornamento dello stato di un modello tramite un metodo *storeModel()*. Esso prende in input i dati dello stato del modello serializzati e la Thing Description da cui é stata generata. Ad ogni scrittura, si ha una lettura per la verifica del corretto inserimento dei dati.

2. Lettura dei modelli salvati, tale operazione viene eseguita allo startup del server Flask per poter ricaricare i modelli del Learning Module e rigenerare le instanze di *ExposedThing* all'interno del *TwinService*. Oppure per visualizzare lo stato aggiornato del modello. Tale operazione reinstanzia i modelli e ricarica i dati delle variabili di istanza dell'oggetto *Optimizer*, in modo da garantire una corretta procedura di inferenza delle proprietá o degli effetti di un invocazione di un azione, senza riaddestrare i modelli.

3. Cancellazione dei modelli tramite un metodo *reset()*. Tale metodo cancella gli stati dei modelli precedentemente salvati sull'instanza Redis. Ció permette di ristabilire uno stato iniziale vuoto sulla persistenza dei dati.

4.2.7 Twin Module

Il Twin Module rappresenta una delle componenti principali di tutto il framework. Esso é responsabile di costruire un *ExposedThing* a partire da un oggetto *ThingDescription* con interaction affordance simili a quelli della WT di riferimento. In altri termini, si costruisce una Thing Description con le medesime proprietá e azioni della TD del WT originale, in modo da rendere replicabile ogni interaction affordance della WT all'interno della WDT. Inoltre, ogni TD di una WDT, contiene una proprietá *modelId* accessibile sia in lettura che in scrittura. Inoltre, la proprietá *time* viene definita anche con accessi in scrittura, questo perché il WDT non ha vincoli temporali e la lettura dei valori delle proprietá sono dipendenti dall'inferenza del modello di ottimizzazione nel Learning Module. Il modelId esprime un valore numerico corrispondente all'identificativo del modello salvato all'interno del Learning Module. Come per il Thing Module, anche il Twin Module espone una *ExposedThing*, la principale differenza risiede nell'implementazione delle funzioni legate ai vari *handlers* in lettura e per l'invocazione di azioni. Nel caso del Twin Module, tali handlers utilizzano la libreria *node-fetch* per poter invocare delle REST API all'interno del Core Module ed eseguire le operazioni sul modello del Learning Module.

```
1  this.thing.setPropertyReadHandler("PROP-NAME", async (options: any) => {
2            const time = this.time
3            const modelId = this.modelId
4            return fetch("CORE-ENDPOINT" + "/readPropertyModel", {
5                method: 'POST',
6                body: JSON.stringify({ time: time, prop: "PROP-NAME", id:
                     modelId }),
7                headers: {
8                    'Content-Type': 'application/json',
9                }
10           }).then((res: any) => res.json())
11               .then((payload: any) => {
12                   PROP = payload.y_future
13                   return PROP
14               })
15       });
```

Listing 4.8: Aggiunta di un handler per la lettura di una proprietá di un WDT.

4.2.8 Core Module

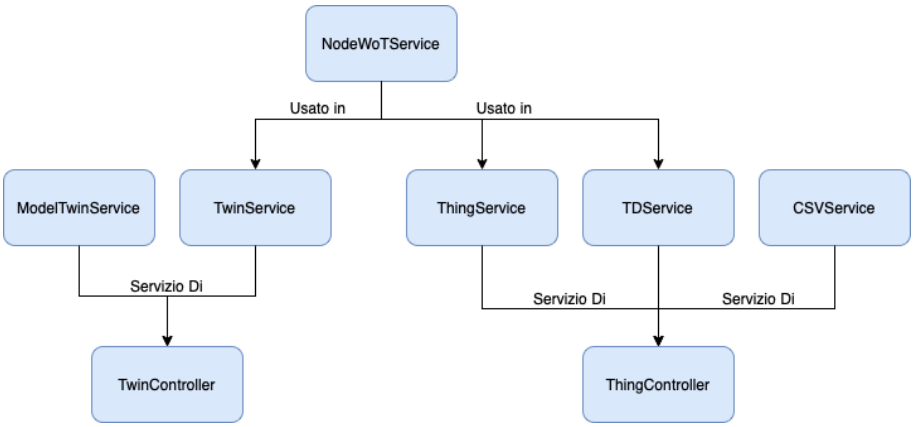

Figura 4.6: Controller e Service implementati nella nuova versione del framework.

Come descritto nella sottosezione 3.7.2, il Core Module raffigura l'unitá principale per la comunicazione tra i moduli del sistema. Partendo dalle funzionalitá descritte nel capitolo precedente, andremo a descrivere come esse siano state implementate nel framework. Il Core Module é implementato in NestJS. Questo permette di definire le funzionalitá in relazione ad API di oggetti di tipo *Controller* e gestirne le procedure attraverso vari *Service*.

Gestione Thing Descriptions Per poter caricare le informazioni relative alle Thing Description visibili al sistema, il Core Module fa affidamento ad un servizio *TDService* in grado di:

1. Prelevare tutte le Thing Description visibili, e conosciute dal sistema, attraverso un metodo *fetchAvailableTDs()*

2. Restituire un oggetto *ThingDescription* a secondo del valore di riferimento dato in input. Tali valori sono univoci e vengono associati all'URL della Thing Description. In seguito, con l'utilizzo di un metodo *fetchTD()* di *NodeWoTService*, si ha la possibilitá di prelevare l'instanza, secondo la struttura fornita da *node-wot*.

3. Selezionare una Thing Description, in relazione all'input ricevuto dalla Dashboard Module e salvarla in una variabile di istanza del servizio

tramite il metodo *switchTD()*. Il riferimento della Thing Description da selezionare é dato in input.

4. Prelevare l'instanza della Thing Description corrente tramite un metodo *fetchTD()*

Se *TDService* gestisce le Thing Description, le instanze delle *ConsumedThing*, generate dal TD selezionato, vengono gestite all'interno del servizio *ThingService*.

Interazione con il Thing Module Per interagire con il Thing Module, ed effettuare operazioni sulle proprietá o invocare azione, si genera una $ConsumedThing$ tramite un servizio condiviso tra i controller di NestJS: $NodeWoTService$. Tale servizio viene inizializzato all'interno delle funzionalitá principali dei controller, per poter effettuare alcune operazioni in relazione al Web of Things e alle interfacce esposte da *node-wot*. Nello specifico, esso instanzia un oggetto $Servient$ per poter prelevare il contesto WoT utilizzato per:

1. Generare un oggetto $ConsumedThing$ in relazione ad un oggetto *ThingDescription* dato dal metodo $consumeFromTD()$.

2. Generare un oggetto $ConsumedThing$ a partire dall'URL della ThingDescription attraverso il metodo $consumeFromURL()$

3. Prelevare l'oggetto Thing Description dato un url in input, attraverso il metodo *fetchTD()*

4. Prelevare i nomi delle proprietá e delle azioni di una $ThingDescription$ con il metodo $getPropertiesNames()$ e $getActionsNames()$

Le interazioni con la Web Thing avvengono all'interno di una servizio *ThingService*, legato al *ThingController* per l'interazione tra il Thing Module e il Core Module. All'interno di tale servizio, si ha la possibilitá di inizializzare una $ConsumedThing$, ed utilizzarla per le varie interaction affordance descritte dall'oggetto $ThingDescription$ annesso. Nello specifico, *ThingService* provvede metodi per alcune funzionalitá in stretta correlazione tra la Dashboard Module e la Thing Module:

1. *consumeByUrl()* e *consumeByTD()* sono metodi wrapper per invocare i rispettivi metodi del servizio *NodeWoTService* precedentemente descritto.

2. *readAllProperties()* per leggere tutte le proprietá nel tempo corrente della *ConsumedThing* generata.

3. *readProperty()* per leggere una singola proprietá nel tempo corrente della Web Thing.

4. *invokeAction()* per invocare un azione in relazione alle Interaction Affordance della Web Thing.

5. *sendProcedure()* per iniziare una fase di monitoraggio sulla lettura della proprietá di una Web Thing data una frequenza ed un intervallo di tempo in input. Tale chiamata non é bloccante, ció significa che si ha la possibilitá di effettuare altre operazioni in parallelo al campionamento; come, ad esempio, l'invocazione di azioni. La sessione di monitoraggio va ad aggiungersi ai dati raccolti, in un oggetto mantenuto nella sessione del servizio.

6. *reset()* per cancellare tutti i dati raccolti dalle varie sessioni di monitoraggio.

7. *retrievesHistory()* per prelevare i dati di campionamento raccolti.

8. *loadMonitoredData()* per caricare una sessione di monitoraggio, precedentemente salvata su file, all'interno della sessione corrente

Interazioni con la WT Dashboard Per interagire con la WT dashboard, si ha un controller *ThingController* in grado di definire delle API per l'interazione tra il Dashboard Module e gli altri moduli. Abbiamo giá visto nel paragrafo precedente le varie interazioni con il Thing Module che, peró, non sono direttamente accessibile al Dashboard Module se non tramite il*ThingController*. Esso infatti espone una serie di API:

1. */fetchTD*: Per invocare il metodo *fetchTD()* del *TDService*.

2. */fetchAvailableTD*: Per invocare il metodo *fetchAvailableTD()* del *TDService* e restituire la lista degli oggetti *ThingDescription* alla *MonitorFormComponent* in front-end.

3. */sendProcedure*: Per acquisire i metadati per una sessione di monitoraggio e avviarla all'interno di *ThingService*.

4. */readProperty* per leggere una proprietá della WT corrente dato il nome e le opzioni in input.

5. */readProperties* per leggere tutte le proprietá della WT al tempo corrente.

6. */invokeAction* per invocare un azione sulla WT consumata dal *ThingService*

7. */switchTD* Per invocare il metodo *switchTD()* di *TDService*.

8. */reset* Per cancellare i dati di sessione relativi alle attivitá di monitoraggio.

9. */setMonitorData* Per caricare una sessione di dati di campionamento precedentemente salvata su file.

10. */learningFromData* Definisce un metodo che preleva i dati di monitoraggio dalla sessione di *ThingService*, la converte in formato .csv tramite un servizio *CSVService* e, insieme al titolo della Thing Description della WT analizzata, invocano un metodo */learn* del server Flask del Learning Module. Come descritto nella sottosezione 4.2.5, tale metodo genera un nuovo modello che verrá successivamente salvato all'interno di un database Redis attraverso un metodo *saveModel()* di *ModelTwinService* e all'interno della sessione di *TwinService* di NestJS per l'interazione con il Twin Module.

Interazioni con il Twin Module All'interno del Core Module, il servizio dedicato all'interazione con il Twin Module é *TwinService*. Se *ThingService* gestiva l'interazione con la Thing Module, *TwinService* fa lo stesso con il Twin Module. Esso permette di invocare le varie operazioni di lettura e scrittura verso le proprietá di uno dei Twin mantenuti dal sistema, invocare azioni e gestirne gli stati, e distribuzione dei valori delle proprietá nel tempo. Nello specifico, esso espone vari metodi:

1. *generate()*: Per la generazione di un *ConsumedThing* e dei dati relativi all'instanza del TD associato all'interno di una collezione mantenuta come variabile di istanza del *TwinService*.

2. *generateFromRedis()*: Per la generazione dei modelli allo startup del server Flask. Questa funzione prende in input i nomi delle Thing Description delle Thing, a cui fanno riferimento i modelli e, partendo da tali informazioni, generano dei *ConsumedThing* con il ruolo di WDT. Tale metodo é invocato dalla API */startup* del server Flask per sincronizzare le instanze dei modelli con i relativi WDT visualizzabili in Dashboard.

3. *getTDs()*: Per restituire la collezione delle Thing Description che hanno almeno un WDT associato.

4. *readProperties()*: Che dato l'id di un WDT e una collezione di timestamp in input, restituisce la lista dei valori di ogni proprietá del WDT agli istanti specificati dal *time*.

5. *readProperty()*: Dato l'id di un WDT, una serie di timestamp e il nome di una proprietá in input, predice i valori della proprietá, utilizzando il modello del WDT selezionato.

6. *invokeAction()*: Dato un id di un WDT, il nome dell'azione, il tempo e i dati di input. Esegue l'azione con il tempo e i parametri forniti.

7. *getPropertiesData()*: Fornito un id in input, definisce la lista delle proprietá del WDT avente tali id, se esiste.

8. *getTwinRef()*: Restituisce una coppia chiave-valore, in cui le chiavi sono gli id dei WDT, i valori sono i riferimenti alle Thing Description su cui si basano.

9. *getTwinTDById()*: Restituisce un oggetto *ThingDescription* di un WDT dato il suo id in input.

10. *parseActionType()*: Forniti l'id del WDT e l'azione del modello di ottimizzazione con i relativi stati delle proprietá all'istanze dell'invocazione, definisce, tramite un analisi delle interaction affordance del Thing Description del WDT, quale azione viene eseguita e con che input, restituendone il nome dell'azione, il timestamp di esecuzione e i valori relativi ai vari input dell'azione. Questo metodo viene utilizzato poiché i dati delle azioni nel modello del Learning Module sono definiti

come una lista di informazione. In tale lista si trova il tipo di operazione (osservazione o azione) e una lista di valori che rappresentano uno snapshot dello stato delle proprietá all'istante di esecuzione.

11. *parseStates()*: Fornisce la lista delle informazioni utili per la generazione degli stati e dei nodi del diagramma di stato descritto nella sottosezione 4.2.2. Tale metodo prende in input la lista degli id dei WDT, le liste delle loro azioni e le proprietá, che andranno a definire lo stato di un nodo, il tempo limite e la frequenza di scansionamento delle transazioni.

12. *formatActionTable()*: Metodo per definire i dati delle righe della tabella delle azioni della WDT Dashboard (4.2.2). Essa prende in input l'id del WDT selezionato e la lista delle azioni del suo modello.

Una osservazione da fare é legata all'implementazione delle funzionalitá di *TwinService*. Alcuni metodi sono strettamente correlati con le funzionalitá definite nel *ThingService*. Nello specifico, le funzioni che effettuano operazioni sulle varie Interaction Affordance delle WT e delle WDT, seguono una medesima procedura. Questo ci fa capire che, dal lato di un consumer, l'interazione con una WDT o con una WT varia nelle funzionalitá e non nelle modalitá di interazione. Inoltre, il Core Module interagisce con il Twin Module anche attraverso alcune API di un controller *TwinController*:

1. */invokeActionModel*: Richiamato all'interno delle funzioni degli handler delle azioni del Twin Module, tale API fa riferimento all'invocazione di un modello del Learning Module, per inserire l'azione invocata nella lista delle azioni del modello, ed utilizzarlo per l'inferenza sulla distribuzione dei dati delle proprietá; andando a definire come l'azione eseguita possa influenzare l'andamento dei valori nell'intervallo di tempi futuri.

2. */readPropertyModel*: Richiamato all'interno delle funzioni degli handler della lettura delle propritá del Twin Module, tale API fa riferimento all'invocazione del modello del Learning Module per l'inferenza su una proprietá. Nello specifico, prende in input l'id del WDT, per avere una corretta associazione con il suo modello nel Learning Module, il nome della proprietá e il timestamp della lettura. Tale API andrá ad effettuare un inferenza sul modello per ricavare lo stato in relazione al timestamp indicato, restituendone il valore in output.

Interazione con la WDT Dashboard Le interazioni con la dashboard dei WDT viene definita all'interno del controller *TwinController* del Core Module. Esso comunica con il Twin Module attraverso l'invocazione dei metodi di *TwinService* o API, secondo i metodi descritti nel paragrafo precedente. Inoltre, gestisce le API per la comunicazione con la WDT Dashboard. Nello specifico, le REST API esposte sono:

1. */getTwinTDs*: Restituisce una collezione di *ThingDescription* in cui, per ogni elemento, é instanziato almeno un WDT all'interno del *TwinService*.

2. */readPropertiesTwin*: Invoca il metodo di *readProperties()* del *TwinService* per poter leggere le proprietá di un WDT attraverso il Twin Module, dando in input l'id del WDT e una lista di timestamp.

3. */invokeActionTwin*: Richiamando il metodo *invokeAction()* di *TwinService*, dando in input l'id del WDT, il nome dell'azione, i timestamps di invocazione e i dati di input, esegue una serie di azioni del WDT selezionato attraverso gli handler delle azioni, definiti nel Twin Module.

4. */readPropertyTwin*: Richiamando il metodo *readProperty()* di *TwinService*, dando sempre in input l'id del WDT, il nome della proprietá e una serie di timestamps della lettura, esegue una serie di letture sul WDT attraverso gli handlers delle proprietá, definiti nel Twin Module.

5. */getDifProperties, /getAlgProperties*: Dato l'id del WDT in input, restituiscono la lista delle proprietá differenziali e algebriche secondo la classificazione effettuata nel Learning Module.

6. */getActionsList*: Dato l'id del WDT, restituisce la lista delle azioni registrate all'interno del modello del Learning Module.

7. */getTwins*: Restituisce una collezione chiave-valore secondo la struttura dell'output di *getTwinsRef()* di *TwinService*, precedentemente analizzata.

8. */getStatesInfo*: Definisce la lista degli stati di una serie di WDT per la generazione di un diagramma di stato. Prende in input la lista degli id dei WDT selezionati, le proprietá che andranno a formare uno stato del WDT, la frequenza e il tempo limite di analisi dei dati. Esegue una

serie di chiamate alla funzione legata alla API */getActions* del Learning Module per prelevare le informazioni sulla lista delle azioni dei modelli dei WDT selezionati. Successivamente, esegue l'invocazione del metodo *parseStates()* del *TwinService*, andando a restituire le informazione degli stati dei vari WDT dati in input.

9. *fillActionTwinTable*: Esegue una procedura simile a */getStatesInfo*, con la differenza che l'id del WDT fornito é unico. Invoca il metodo *formatActionsTable* di *TwinService*, fornendo l'id e la lista delle azioni del modello in input. Restituisce la lista delle righe della tabella da visualizzare sulla WDT Dashboard.

10. */deleteAction*: Dato l'id di un WDT, e l'indice dell'azione da dover eliminare, richiama l'API */deleteAction*, del server Flask del Learning Module, per cancellare l'azione in input. Restituisce un booleano in relazione all'esito dell'operazione.

4.3 Caricamento dei WDT dal Redis Store

Nella sezione 3.7.2 abbiamo introdotto le funzionalitá in relazione al Redis Store implementate secondo i metodi definiti in 4.2.6. Partendo da tali informazioni, possiamo definire la procedura di caricamento dei WDT salvati sul database Redis, eseguita in fase di startup. Durante la messa in esecuzione dei container *Docker*, si ha la possibilitá di invocare una funzione di startup per inizializzare la sessione dei WDT del sistema. I dati da caricare sono immagazzinati in maniera persistente nel Redis Store. L'implementazione della procedura di caricamento dei WDT puó essere sintetizzata nei seguenti punti:

1. Esecuzione di un invocazione di una funzione *startup()* nel server Flask del Learning Module. Tale funzione richiede il caricamento dei modelli al Core Module attraverso una chiamata all'API */loadModels* del *TwinController*.

2. Il TwinController richiama il servizio *ModelTwinService* [4.2.6] per invocare una chiamata al metodo *loadModels()*.

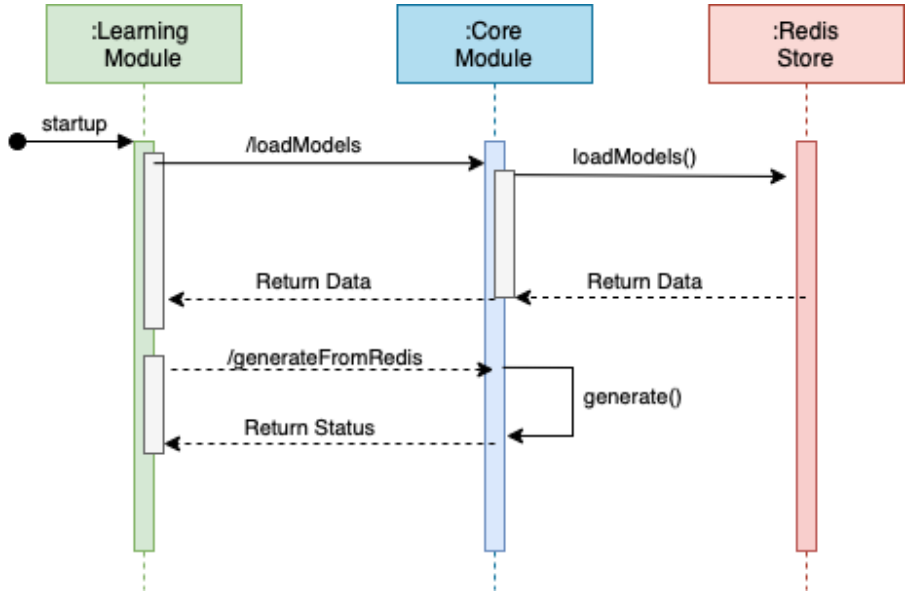

Figura 4.7: Sequence Diagram per il caricamento dei WDT e dei modelli dal Redis Store

3. Il *TwinService* effettuerá una serie di operazioni GET per prelevare i valori legati agli ID dei modelli. Tali operazioni vengono effettuate attraverso un instanza di *CacheModule*.

4. I dati prelevati vengono dati in risposta alla richiesta del Learning Module. Successivamente, i dati verranno salvati in sessione per un utilizzo ottimale dei modelli.

5. Infine, il Learning Module invoca una chiamata alla API */generateFromRedis* del *TwinController*, nel Core Module. Tale chiamata effettua un fetch delle Thing Description dei modelli caricati e genererá, tramite la funzione *generate()* del TwinService, le instanze di *ConsumedThing*, rappresentanti i WDT, per l'interfacciamento verso le funzionalitá richiamate dalla WDT Dashboard. In risposta alla chiamata, si avrá lo stato dell'esito dell'operazione in JSON.

4.4 Generazione di un nuovo WDT

Presentando tutti i moduli dell'architettura, e come essi sono stati implementati, possiamo definire ora la procedura implementata per la generazione

del WDT. Essa puó essere riassunta nei seguenti punti:

1. L'utente finale clicca su un bottone "Learning from Data" all'interno della Dashboard della Web Thing.

2. La componente *MonitorFormComponent*, cattura l'evento ed invoca il metodo *learningFromData()* che, al suo interno, invoca un metodo omonimo del *SettingTDService*.

3. Una volta formattato il payload della richiesta HTTP, il metodo *learningFromData()* di SettingTDService invoca il servizio *NetworkService* per generare, tramite una funzione omonima, una richiesta HTTP verso l'API /learningFromData del Core Module.

4. Il Core Module preleva i dati di monitoraggio della sessione attraverso il metodo *retrievesHistory()* del *MonitorControllerService*, ne formatta i dati e genera un dataset in formato .csv, usando il metodo *generateCSV()* del *CSVService*.

5. Successivamente, invoca il server Flask del Learning Module, tramite l'API */learn*, andando a fornire i dati formattati, con il titolo della Thing Description della Web Thing del *ThingService*, in input.

6. Il metodo *learn()* del server Flask invoca il metodo *init()* su una nuova instanza della classe *Optimizer*. I dati di input sono quelli della richiesta del punto precedente. Il risultato dell'invocazione definisce un nuovo modello addestrato, salvato nella sessione del server Flask ed inviato nella risposta alla chiamata */learn*. Contemporaneamente, si invoca una chiamata API a */storeModel* del *TwinController* del Core Module per salvare lo stato del nuovo modello su Redis.

7. Il Core Module riceve i dati di risposta dal server Flask e inoltra i dati del modello al *TwinService* tramite una chiamata POST all'API */generate* del *TwinController*.

8. Il TwinController esegue una chiamata */fetchTD* del *ThingController* per prelevare le informazioni sulla Thing Description su cui il modello del Learning Module é basato. Successivamente, genera le informazioni del Twin tramite il metodo *generate()* di *TwinService*, dando in input i dati della Thing Description e del modello del Learning Module.

9. Infine, si restituisce lo stato di generazione del WDT, andando a fornire un valore sull'esito della generazione.

10. La risposta arriva alla WT Dashboard e, a secondo del valore ricevuto, stampa un messaggio di successo o di errore attraverso il servizio *EventBusService*.

L'intera procedura puó essere riassunta nel seguente Sequence Diagram:

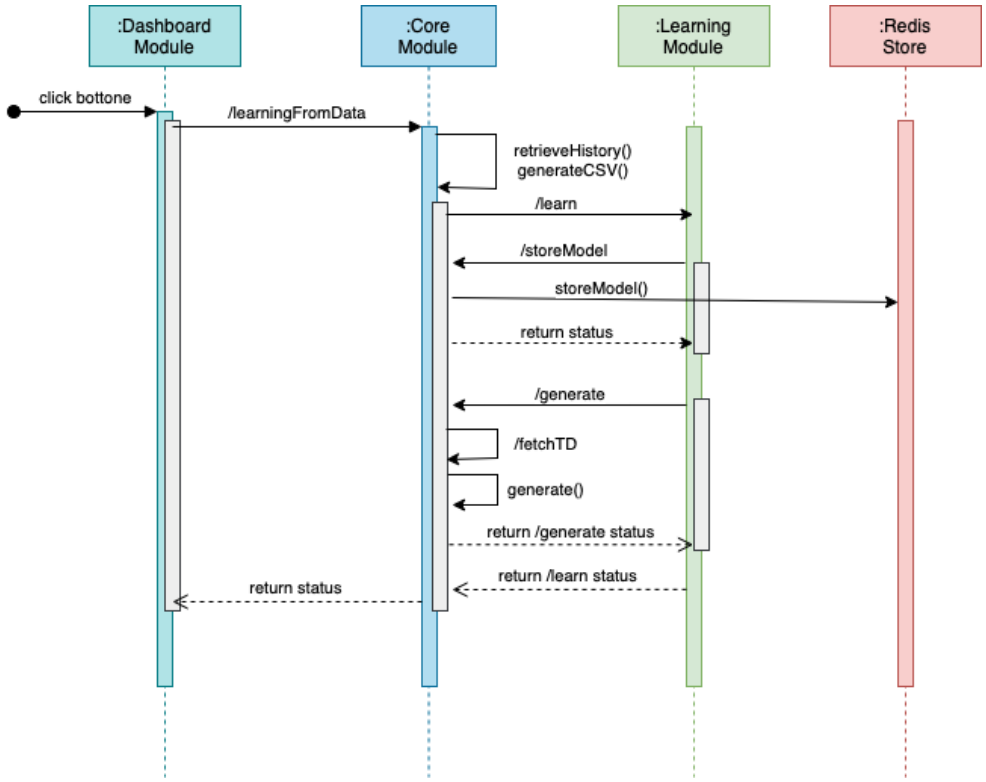

Figura 4.8: Sequence Diagram per la generazione di un nuovo WDT

Capitolo 5

Validazioni

5.1 Caso d'uso: Relativistic Room

Il caso d'uso é preso dalla versione precedente del framework. Introdurremo brevemene la struttura logica dei suoi modelli, per poter rendere noto l'ambiente di sperimentazione. L'intero caso d'uso é simulato, con un implementazione all'interno del Simulation Module, richiamato all'interno delle funzioni degli handler sulle proprietá della WT. L'ambiente di simulazione comprende due stanze con due tipi di attuatori:

1. **Sistema di Riscaldamento (Heater)**: In grado di aumentare la temperatura. Puó assumere due valori: *false* se spento e *true* se acceso ad un tempo t. Nell'ambiente simulato, ogni stanza possiede un proprio riscaldamento.

2. **Refrigeratore (Cooler)**: In grado di diminuire la temperatura. É presente solamente nella prima stanza.

Le variabili T_A e T_B rappresentano la temperatura della prima e della seconda stanza, lo stato del refrigeratore $C(t)$ influenza solamente la prima stanza, mentre esistono due potenze del riscaldamento centralizzato H_A e H_B legate ognuna ad una stanza. Lo stato del refrigeratore é rappresentato da $c(t)$, il suo valore é definito dall'utente e puó variare da *0* a *8*. Lo stato del riscaldamento é centralizzato, il valore é definito attraverso la variabile $h(t)$ definito dall'utente; il suo valore puó essere *1* se acceso, *0* se spento. Le azioni per modificare il valore del refrigeratore e del riscaldamento centralizzato sono: *toggleCooler* e *toogleHeater*. Le stanze sono comunicanti attraverso un muro,

rendendo i valori delle temperature mutuamente influenzabili. Il modello del simulatore é definito secondo le seguenti equazioni differenziali:

$$\hat{T}_A(t) = p_1(p_2(T_{out}(t) - T_A(t)) + H_A(t) - C(t) + p_3(T_B(t) - T_A(t)))$$
$$\hat{T}_B(t) = p_4(p_5(T_{out}(t) - T_B(t)) + H_B(t) + p_3(T_B(t) - T_A(t)))$$
$$\hat{H}_A(t) = p_6(p_7 h(t) - H_A(t))$$
$$\hat{H}_B(t) = p_6(p_7 h(t) - H_B(t))$$
$$\hat{C}(t) = p_9 c(t)$$

Il simulatore segue l'inerzia termica sul sistema di riscaldamento. Per aumentare la variabilitá ed evitare che il WDT segua perfettamente l'andamento della simulazione, i modelli sul DT del WT, da cui é generato, avrá l'inerzia termina sul sistema di raffreddamento. Ció modificherá le equazioni differenziali sulla potenza legata al sistema di riscaldamento e del raffreddamento nel seguente modo:

$$\hat{H}_A(t) = p'_7 h(t))$$
$$\hat{H}_B(t) = p'_8 h(t))$$
$$\hat{C}(t) = p_9(p_{10} c(t) - C(t))$$

I parametri p della simulazione variano le caratteristiche dell'andamento dei valori, mentre le rispettive variabili nel modello del WDT avranno lo scopo di determinare il corretto andamento della simulazione, attraverso la fase di addestramento e l'applicazione della minimizzazione della Least Squares, in relazione ai parametri ottimali risultanti dalle soluzioni della ODE (Sezione 3.5).

5.2 Esperimenti

La sperimentazione sul framework avrá come fine quello di verificare la correttezza delle sue funzionalitá. Nello specifico, si andrá a:

1. Generare e confrontare i modelli del WDT, in relazione a differenti tempi di addestramento, sull'andamento dei valori di una WT di riferimento.

2. Si analizzeranno le variazioni di stato dei differenti WDT in relazione a differenti threshold per uno specifico intervallo; in mdo da verificare la validitá delle informazioni mostrare nei diagrammi.

5.2.1 Settings

Il sistema viene eseguito interamente in locale, i vari moduli sono salvati all'interno di container Docker. La macchina utilizzata per le sperimentazioni ha le seguenti caratteristiche hardware:

- Processore Intel Core i7, 2.2 Ghz
- Memoria RAM 16GB, 2400 Mhz DDR4
- Scheda Grafica Radeon Pro 55X 4GB
- Scheda Grafica Integrata Intel UHD Graphics 630 1536 MB
- Sistema Operativo MacOS Monterey

5.2.2 Risultati

In seguito agli esperimenti eseguiti secondo le modalitá definite nella sezione 5.2, andiamo ora a mostrare i risultati raggiunti.

Training Modelli WDT

Di seguito, mostriamo tre differenti coppie di grafici che illustrano l'andamento della temperatura predetta dal WDT in relazione alla variazione dei valori simulati nelle stanze del WT.Una linea verticale definirá il punto di produzione del WDT.

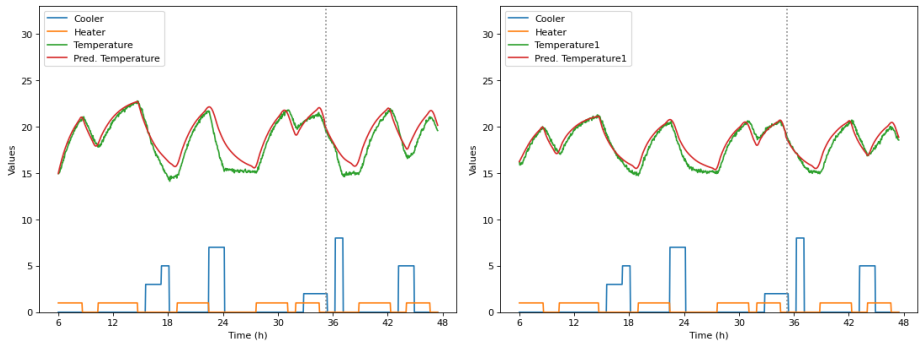

Figura 5.1: Generazione di un WDT partendo dal 70% dei dati campionati per le temperature delle due stanze.

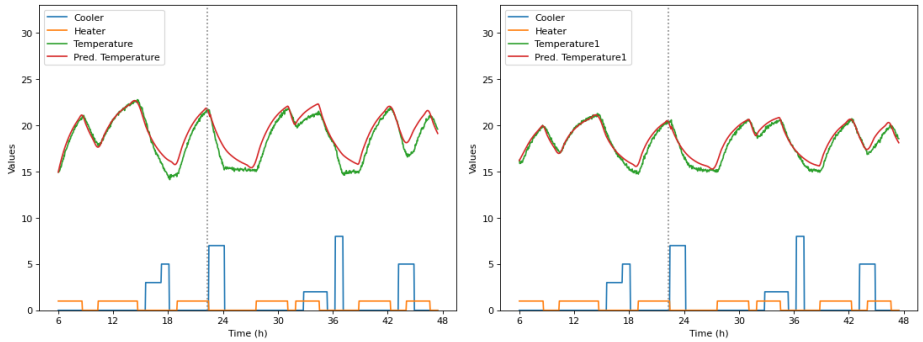

Figura 5.2: Generazione di un WDT partendo dal 40% dei dati campionati per le temperature delle due stanze.

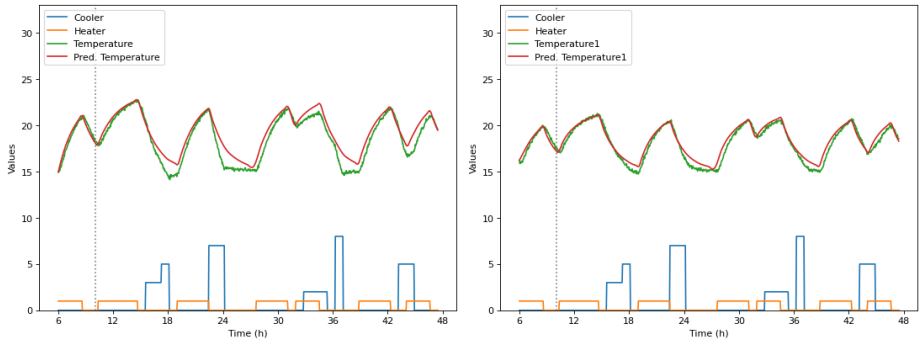

Figura 5.3: Generazione di un WDT partendo dal 10% dei dati campionati per le temperature delle due stanze.

Da come si puó osservare, l'andamento delle temperature predette dai WDT

non varia notevolmente al variare della quantitá di dati utilizzati per l'addestramento. Questo comportamento é dovuto alla presenza dei valori *guess* introdotti in sezione 3.4. Possiamo quindi affermare che, a secondo delle conoscenze delle variabili di ambiente, si ha la possibilitá di poter utilizzare anche una quantitá minima di dati di addestramento, in modo da poter generare un WDT efficiente.

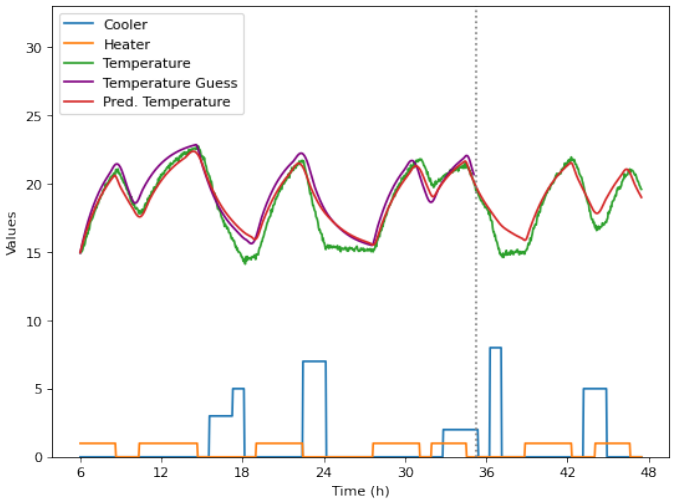

Figura 5.4: Andamento dei valori secondo la predizione del WDT, i valori reali del WT e i valori guess sulla temperatura basati sul modello matematico del TD.

Diagramma di Stato

Infine, possiamo definire come varia la generazione degli stati in relazione ad alcuni threshold inseriti per un intervallo di tempo prefissato dall'utente. In altri termini, da tali funzionalitá possiamo prevedere se e quando un valore superi una determinata soglia per alcuni valori delle sue proprietá. In Figura

Proprietá	**min.**	**max.**
Temperatura	17.0	22.0
Cooler	0	6

Tabella 5.1: Threshold per il diagramma di stato nel diagramma di stato 5.5

Azione	Input	Timestamp
toggleHeater	Heater: 1	18000s
toggleHeater	Heater: 0	36000s
toggleCooler	Cooler: 3	39600s

Tabella 5.2: Azioni da eseguire in 5.5

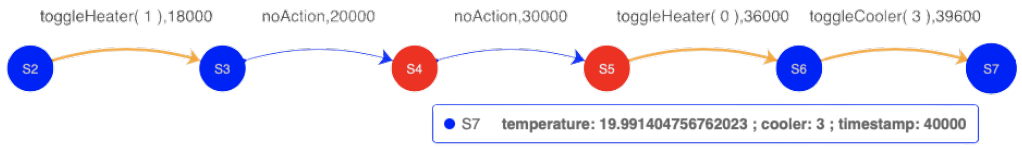

Figura 5.5: Sottografo del diagramma di stato del WDT per l'esecuzione delle azioni in 5.2 con threshold specificati in 5.1

5.5 possiamo analizzare l'andamento dei valori di uno stato formato dalle proprietá di temperatura e del cooler. Alla simulazione dell'esecuzione delle azioni elencate in tabella 5.2, il cambiamento di stato é definito in relazione ai valori dei threshold in tabella 5.1. Partendo da uno stato *S2* si attiva il sistem di riscaldamento. All'attivazione, nello stato *S2* con timestamp *20000*, non vi é variazione sullo stato del DWT in relazione ai vincoli, mantened $17.0 \leq t \leq 22.0$. Successivamente, all'istante *30000* si verifica un cambiamento di stato dovuto ad un aumento della temperatura causata dall'attivazione del sistema di riscaldamento. Essendo il nodo *S4* di colore rosso, non avendo modificato lo stato del Cooler, si avrá la violazione del limite superiore del threshold della temperatura, ossia si avrá il caso di $t \geq 22.0$. In seguito allo spegnimento del sistema i riscaldamento al tempo *36000*, si ritorna nel range del threshold allo stato *S6*. Infine, modificando il valore del sistema di raffreddamento con valore 3 all'istante *39600*, si mantiene lo stato normale sui threshold di temperatura e cooler fino allo stato *S7* con timestamp *40000*. La generazione degli stati viene effettuata in relazione al cambiamento di stato del WDT nel tempo o, se si mantiene uno stato di un tipo costante, secondo una frequenza di monitoraggio definita in input. Nell'esempio riportato, tale frequenza é impostata ad ogni *10000* secondi.

Capitolo 6

Conclusioni e Lavori Futuri

Nel presente elaborato di tesi sono state presentate soluzioni per l'implementazione di un sistema automatico per la generazione di uno o piú Digital Twin, all'interno di un ambiente Web of Things. Si é considerato di implementare tale sistema in una architettura a microservizi in grado di gestire una raccolta dati su Web Thing basate su dispositivi reali o simulati e, una volta acquisiti i dati, utilizzarli per la generazione di un Web Digital Twin (WDT). Tale lavoro si é reso necessario in quanto la generazione di Digital Twin ad oggi non raffigura un processo semplice ed automatico. Si sono studiati i meccanismi propri del W3C WoT e della sua implementazione attraverso il framework *node-wot* per poter implementare moduli capaci di generare e gestire l'interazione con i Web Digital Twin prodotti. Si sono sfruttate le caratteristiche semantiche fornite dalle WoT Thing Description, per definire un modello matematico capace di poter determinare lo stato di una proprietá al variare del tempo. L'idea alla base del progetto é proprio quella di definire un meccanismo che sfrutti proprio le informazioni semantiche di una Thing Description per poter generare la struttura logica di un Web Digital Twin. La realizzazione del presente progetto si basa sullo studio di un applicazione giá esistente [29], alla quale sono stati aggiunti nuovi requisiti funzionali. Nello specifico, si é partiti dallo studio dell'applicazione esistente creando una lista di funzionalitá mancanti e reimplementando alcuni servizi per renderli modulari per la supportabilitá verso differenti casi d'uso. Successivamente, si sono definite le specifiche da dover implementare in funzione agli obiettivi del progetto. Da tale precondizione, si é voluto estendere il framework verso funzionalitá che consentano ad utenti e sviluppatori di analizzare e monitorare dispositivi fisici in un ambiente WoT, per poi generare e gestire multipli

Web Digital Twin per l'analisi del comportamento e dei cambi di stato delle proprietá di una Web Thing. Tale framework é capace di definire un ambiente di gestione completo verso Web Thing e Web Digital Twin, eseguendo azioni che possano cambiare lo stato del sistema analizzato, monitorare i cambiamenti attraverso un Web Digital Twin associato, e trarre conclusioni sul comportamento di uno o piú dispositivi. Successivamente, si é adottato un approccio modulare per poter rendere il framework accessibile verso l'interfacciamento a piú Digital Web Twin aventi caratteristiche differenti. Un punto debole di questo framework é dovuto alla mancata possibilitá di poter gestire l'interazione tra Web Thing e una o piú Web Digital Twin all'interno di in un ambiente real-time. Possibili lavori futuri potrebbero interfacciarsi sull'adattabilitá dell'architettura verso un ambiente event-driven, cosi da poter gestire le varie letture sui dati dei dispositivi di una Web Thing con un analisi definita ed effettuata all'interno di una Web Digital Twin. Infine, il progetto potrebbe essere arricchito andando a definire delle funzionalitá in grado di poter implementare un meccanismo di integrazione tra multiple Web Thing. In altri termini, si potrebbero sviluppare soluzioni in grado di generare Web Digital Twin in differenti livelli di astrazione, sfruttando le potenzialitá di gerarchizzazione giá presente nelle Thing Description del W3C WoT.

Bibliografia

[1] Hammi, Badis & Khatoun, Rida & Zeadally, Sherali & Fayad, Achraf & Khoukhi, Lyes. (2017). Internet of Things (IoT) Technologies for Smart Cities. IET Networks. 7. https://doi.org/10.1049/iet-net.2017.0163.

[2] Al-kahtani, M.S.; Khan, F.; Taekeun, W. Application of Internet of Things and Sensors in Healthcare. Sensors2022,22,5738. https://doi.org/10.3390/s22155738

[3] V. C. Jadala, S. K. Pasupuletti, S. H. Raju, S. Kavitha, C. M. H. Sai Bhaba and B. Sreedhar, "Need of Intenet of Things, Industrial IoT, Industry 4.0 and Integration of Cloud for Industrial Revolution," 2021 Innovations in Power and Advanced Computing Technologies (i-PACT), 2021, pp. 1-5, doi: 10.1109/i-PACT52855.2021.9696696.

[4] Kevin Ashton, "That 'Internet of Things' Thing, In the real world, things mater more than ideas, 2009

[5] Energy Consumption Optimization in Internet of Things Applications: Concept and Techniques, Vishal Barot, 2020, http://doi.org/10.13140/RG.2.2.14863.20642

[6] State of IoT 2022: Number of connected IoT devices growing 18% to 14.4 billion globally, Mohammad Hasan, 2022. https://iot-analytics.com/number-connected-iot-devices/

[7] Internet of Things-IOT: Definition, Characteristics, Architecture, Enabling Technologies, Application & Future Challenges, Sunil Patel, Carlos Salazar, 2016, http://doi.org/10.4010/2016.1482

[8] Khare, Shivanjali & Totaro, Michael. (2019). Big Data in IoT. 1-7. https://doi.org/10.1109/ICCCNT45670.2019.8944495.

[9] Dawood, Afrah. (2020). Internet of Things (IoT) and its Applications: A Survey. International Journal of Computer Applications. 175. 975-8887. 10.5120/ijca2020919916.

[10] A. Zanella, N. Bui, A. Castellani, L. Vangelista and M. Zorzi, "Internet of Things for Smart Cities," in IEEE Internet of Things Journal, vol. 1, no. 1, pp. 22-32, Feb. 2014, doi: 10.1109/JIOT.2014.2306328.

[11] H. Bhatia, S. N. Panda and D. Nagpal, "Internet of Things and its Applications in Healthcare-A Survey," 2020 8th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), 2020, pp. 305-310, doi: 10.1109/ICRITO48877.2020.9197816.

[12] A. Shrivastava, A. Bhardwaj and N. Hasteer, "IoT in Automobile Sector: State of the Art," 2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence), 2020, pp. 254-259, doi: 10.1109/Confluence47617.2020.9058202.

[13] A. Karmakar, N. Dey, T. Baral, M. Chowdhury and M. Rehan, "Industrial Internet of Things: A Review," 2019 International Conference on Opto-Electronics and Applied Optics (Optronix), 2019, pp. 1-6, doi: 10.1109/OPTRONIX.2019.8862436.

[14] Mughal, Muhammad Jawad & Faisal, Mohammad. (2019). Internet of Things - IoT Interoperability and Challenges. Journal of Mechanics of Continua and Mathematical Sciences. 14. 304-318. doi.org/10.26782/jmcms.2019.08.00025.

[15] M. Kovatsch, R. Matsukura, M. Lagally, T. Kawaguchi, K. Toumura, and K. Kajimoto, "Web of Things (WoT) Architecture," W3C Recommendation, Apr. 2020, https://www.w3.org/TR/wot-architecture/

[16] Web of Things (WoT) Thing Description. Sebastian KÃ¤bisch; Takuki Kamiya; Michael McCool; Victor Charpenay; Matthias Kovatsch. W3C. 30 January 2020. W3C Proposed Recommendation. https://www.w3.org/TR/2020/PR-wot-thing-description-20200130/

[17] JSON-LD 1.1. Gregg Kellogg; Pierre-Antoine Champin; Dave Longley. W3C. 5 March 2020. W3C Candidate Recommendation. https://www.w3.org/TR/json-ld11/

[18] Web of Things (WoT) Binding Templates. Michael Koster; Ege Korkan. W3C. 30 January 2020. W3C Note. https://www.w3.org/TR/2020/NOTE-wot-binding-templates-20200130/

[19] Web of Things (WoT) Scripting API. Zoltan Kis; Daniel Peintner; Johannes Hund; Kazuaki Nimura. W3C. 28 October 2019. W3C Working Draft. https://www.w3.org/TR/wot-scripting-api/

[20] Web of Things (WoT) Security and Privacy Guidelines. ; Michael McCool; Elena Reshetova. W3C. July 2021. https://w3c.github.io/wot-security/

[21] M. Grieves. (2015). Digital Twin: Manufacturing Excellence Through Virtual Factory Replication. Digital Twin White Paper. Accessed: Oct. 16, 2019.

[22] M. Grieves and J. Vickers, Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. Springer, 2017.

[23] Grieves, Michael. (2016). Origins of the Digital Twin Concept. https://doi.org/10.13140/RG.2.2.26367.61609

[24] U.S. Air Force. (2013). Global Horizons Final Report: United States Air Force Global Science and Technology Vision. AF/ST TR 13-01, United States Air Force. https://www.hsdl.org/?view&did=741377

[25] Anis Assad Neto, Elias Ribeiro da Silva, Fernando Deschamps, Edson Pinheiro de Lima, Digital twins in manufacturing: An assessment of key features, Procedia CIRP, 2021, https://doi.org/10.1016/j.procir.2020.05.222.

[26] Digital Twins in the Automotive Industry: The Road toward Physical-Digital Convergence, Dimitrios Piromalis and Antreas Kantaros, 2022, http://doi.org/10.3390/asi5040065

[27] Kim, D.; Yoon, Y.; Lee, J.; Mago, P.J.; Lee, K.; Cho, H. Design and Implementation of Smart Buildings: A Review of Current Research Trend. Energies 2022, 15, 4278. https://doi.org/10.3390/

[28] Hassani, H.; Huang, X.; MacFeely, S. Impactful Digital Twin in the Healthcare Revolution. Big Data Cogn. Comput. 2022, 6, 83. https://doi.org/10.3390/

[29] Luca Sciullo, Angelo Trotta, Federico Montori, Luciano Bononi, Marco Di Felice, Department of Computer Science and Engineering, University of Bologna, Italy Advanced Research Center on Electronic Systems Ercole De Castro, University of Bologna, Italy

[30] Simplified Mathematical Modelling of Dehumidifier and Regenerator of Liquid Desiccant System, Rakesh Kumar and Arun K. Asati, 2014, http://inpressco.com/wpcontent/uploads/2014/03/Paper20557563.pdf

[31] Improved Magnus Form Approximation of Saturation Vapor Pressure, Oleg A. Alduchov1 and Robert E. Eskridge1, 01 Apr 1996 https://doi.org/10.1175/1520-0450(1996)035<0601:IMFAOS>2.0.CO;2

Ringraziamenti

Mi é doveroso dedicare questo spazio del mio elaborato alle persone che hanno contribuito, con il loro instancabile supporto, alla realizzazione dello stesso.

In primis, un ringraziamento speciale al mio relatore Di Marco Di Felice, e il mio correlatore Luca Sciullo per la loro immensa pazienza, per gli indispensabili consigli, per le conoscenze trasmesse durante tutto il percorso di stesura dell'elaborato (sia a livello professionale che interpersonale).

Ringrazio infinitamente i miei genitori che mi hanno sempre sostenuto, appoggiando ogni mia decisione, fin dalla scelta del mio percorso di studi, nei momenti brutti e belli. A chi mi hanno supportato in qualsiasi pensiero, idea, decisione che io abbia preso nella mia vita e che mi danno la possibilitá di continuare a seguire le mie passioni.

Un grazie di cuore a tutti i miei colleghi universitari, che mi hanno dato una mano nei momenti difficili, condiviso momenti felici, aiutato nei momenti critici: Andrea G., Giovanni, Tommaso, Michele, Riccardo, Giacomo, Mauro, Andrea D., Giosué, Davide e tutti gli altri che, in un modo o nell'altro, mi hanno fatto crescere professionalmente e come persona.

Un grazie a Michele, Paolo, Andrea e tutti gli amici di Fisciano che, anche se lontani, rimangono e rimarranno sempre come dei fratelli.

Un grazie agli amici di Bologna, alle balotte serali, ai drink di troppo, agli imprevisti che mi hanno fatto vivere bellissimi momenti.

Per ultimi, ma non meno importanti, ringrazio di cuore gli amici di una

vita, che dopo tanti anni conoscono i miei pregi e i miei difetti e nonostante tutto mi sono sempre vicini, nel particolare ringrazio Lorenzo, Cono, Giovanni, Giannino, Federico, Vincenzo e tutti gli altri (siete troppi, mi annoia scrivere dopo 100 pagine di tesi).

Infine, un ringraziamento speciale ai miei fallimenti che mi hanno fatto capire molto di me, dei miei punti deboli e, a volte, mi hanno fatto crescere (spero in positivo).

Mario Sessa