ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea in Ingegneria e Scienze Informatiche

SVILUPPO DI DIGITAL TWIN UTILIZZANDO IL FRAMEWORK ECLIPSE DITTO

Elaborato in
SISTEMI EMBEDDED E INTERNET-OF-THINGS

Relatore
Prof. ALESSANDRO RICCI

Presentata da
ALESSANDRO BECCI

Anno Accademico 2020 – 2021

Indice

Introduzione

I Digital Twin, o "Gemelli Digitali", sono un concetto che negli ultimi anni si è sviluppato molto all'interno di vari settori, come quello automotive, della sanità, o industriale. Inoltre, è un concetto fondante anche delle Smart City e quindi, della mobilità sostenibile.

La loro concezione e sviluppo è piuttosto recente, e ancora si fatica a trovarne una definizione univoca. È comunque possibile delineare una definizione base valida per tutti gli ambiti: *Un Digital Twin è la rappresentazione virtuale di un oggetto fisico, che riflette tutte le sue proprietà e comportamenti utili all'interno di un contesto.*

Il loro primo utilizzo fu in ambito industriale, nell'ambito del ciclo di vita di un prodotto, ma poi, il concetto è stato traslato in numerosi ambiti, acquisendo significati nuovi. Ad esempio, ora non è strano imbattersi in Digital Twin che non riflettono un vero e proprio oggetto fisico, ma anche attività o processi.

Nella mia tesi, presenterò la storia dei Digital Twin, analizzando gli ambiti che ne hanno più influenzato lo sviluppo, e le proprietà fondamentali che negli anni sono emerse. Inoltre, esplorerò un framework open-source per la loro creazione, sviluppo ed utilizzo: Eclipse Ditto.

Presenterò le sue parti, e il suo funzionamento, ponendo l'enfasi anche sull'integrazione con il Web Of Things, standard proposto W3C per avere interoperabilità a livello applicativo nel mondo dell'IoT e affrontare i problemi di frammentazione.

Dopo aver presentato Eclipse Ditto e il Web Of Things, passerò ad un esempio: il Digital Twin di una serra. Sistemi Smart relativi alle coltivazioni sono ormai una realtà, e anche qui il concetto di Digital Twin calza perfettamente.

Capitolo 1

Panoramica sui Digital Twin

La panoramica si basa sul survey di Minerva, Lee e Crespi [1], che fornisce un quadro complessivo riguardo ai Digital Twin e ai concetti di riferimento che lo caratterizzano. Partirò dalle loro origini, analizzando poi i trend tecnologici che hanno contribuito al loro sviluppo e infine presenterò le proprietà fondamentali che vengono delineate dal documento.

1.1 Origini

Il concetto viene coniato da Micheal Grieves, che lo presenta nel 2003 all'Università del Michigan[2]. Inizialmente riguardava soprattutto l'ambito industriale, e quindi il Digital Twin era la copia tipicamente di un prodotto.

Il concetto si è poi ampliato fino ad essere una nozione applicabile a qualsiasi oggetto fisico, e in linea di principio, anche ad oggetti intangibili fisicamente. Come definizione di partenza, il paper riporta quella data da Haag e Anderl. nel "Digital Twin-Proof of Concept"[3]:

"a Digital Twin is a comprehensive software representation of an individual physical object. It includes the properties, conditions, and behavior(s) of the real-life object through models and data. A Digital Twin is a set of realistic models that can simulate an object's behavior in the deployed environment. The Digital Twin represents and reflects its physical twin and remains its virtual counterpart across the object's entire lifecycle"

Questa definizione esprime le seguenti importanti nozioni:

- Un Digital Twin è la rappresentazione software di un (singolo) oggetto fisico.
- Un Digital Twin contiene le informazioni necessarie atte a rappresentare le proprietà e il comportamento dell'oggetto fisico.

- Un Digital Twin rappresenta l'oggetto fisico in tutto il suo ciclo di vita, e può simularne il comportamento.

1.2 Trend scientifici e tecnologici che hanno influenzato i Digital Twin

Questa sezione discute di come differenti trend abbiano influenzato la definizione corrente di Digital Twin. L'idea comune di replicare attraverso il software degli oggetti fisici resta la stessa per tutti gli ambiti, ma questi differiscono sulle definizioni di proprietà e caratteristiche, applicandole al proprio campo d'utilizzo. È giusto quindi fare una panoramica di come i vari ambiti abbiano sviluppato il concetto di Digital Twin.

1.2.1 Contributi dall'industria manifatturiera

Il concetto di Digital Twin è nato nell'industria manifatturiera, e le motivazioni non sono difficili da trovare. Si pensi ad un macchinario, o ad un sistema di essi: l'unico modo per conoscere le sue proprietà e i suoi comportamenti è quello di metterlo in osservazione diretta da parte di un operatore.

Creando la sua copia digitale è possibile invece compiere analisi e rilevazioni anche da remoto. La rappresentazione digitale deve essere in grado di rappresentare un asset fisico in tutte le parti del suo ciclo di vita: design, prototipazione e testing.

Spesso viene creata la copia di un oggetto prima ancora che l'oggetto esista: in questo modo è possibile eseguire test e simulazioni per orientare al meglio lo sviluppo del prodotto.

Ciclo di vita di un prodotto e Digital Twin

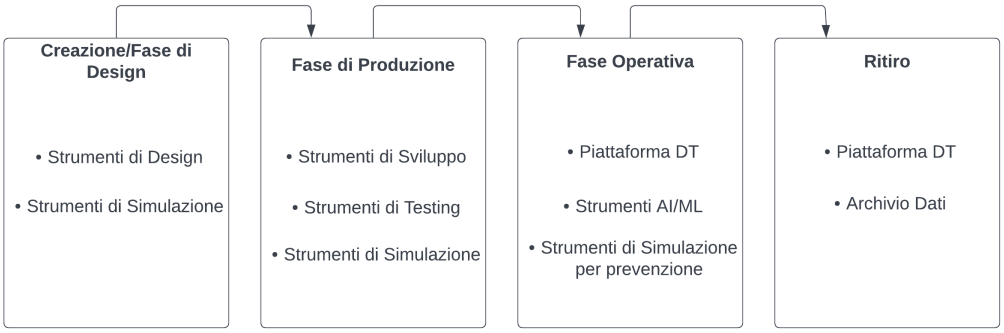

Figura 1.1: Ciclo di vita di un prodotto applicato ai DT

I Digital Twin hanno un impatto importante anche per quanto riguarda il ciclo di vita di un prodotto, come si vede dalla figura 1.1.

Durante la prima fase, quella di design, il prodotto viene concepito e progettato, e il Digital Twin viene utilizzato per compiere simulazioni, allo scopo di compiere scelte sulle sue caratteristiche ed evitare di sviluppare prototipi inutili portando ad uno spreco di denaro e tempo.

Quando il prodotto entra in fase di produzione, vengono utilizzati strumenti di testing per verificare che il prodotto sia conforme alle specifiche, e anche qui vengono utilizzate delle simulazioni: lo scopo è differente a quello nella fase di design, qui le si usano ad esempio per controllare la presenza di malfunzionamenti.

Le simulazioni sono utili anche nella fase operativa del prodotto: il Digital Twin grazie ad esse può sviluppare una propria intelligenza (tramite tecniche come intelligenza artificiale e machine learning) e compiere scelte in base, ad esempio, a situazioni di pericolo. Uno dei vantaggi è proprio quello di poter compiere test e sperimentazioni durante condizioni e ambienti differenti: senza la possibilità di utilizzare una copia digitale, si dovrebbero realizzare numerosi prototipi e le condizioni dovrebbero essere sempre "reali", senza avere la possibilità di simularle.

Allo scopo di eseguire queste simulazioni, è necessario mantenere le informazioni del Digital Twin su una qualche piattaforma, ed è necessario anche mantenerle aggiornate: di questo parlerò nei prossimi capitoli, quelli legati alle proprietà fondamentali di un DT.

Anche quando il ciclo di vita di un prodotto è terminato, è comunque utile mantenere i dati della copia digitale: potrebbero essere utili in futuro per sviluppare prodotti dello stesso tipo.

1.2.2 Contributi dalle tecniche di realtà aumentata e virtuale

Si intende come realtà aumentata la possibilità di aumentare le funzionalità e i dati a disposizione di un oggetto fisico, mentre la realtà virtuale si pone come obiettivo quello di creare un ambiente virtuale completo con cui un utente può interagire.

Le sue applicazioni riguardano ad esempio il cinema, la medicina, ma sono molti gli ambiti in cui queste tecnologie vengono applicate o potranno esserlo in futuro.

Metaverso

Il metaverso è una delle più importanti applicazioni delle tecniche di realtà aumentata e di realtà virtuale, che negli ultimi anni ha attratto le più grandi aziende tecnologiche del mondo(Apple, Microsoft, Meta).

Esso può essere inteso anche come una evoluzione del web: si pensi ad un mondo virtuale in cui si possono eseguire attività comuni, affini alla vita di tutti i giorni, ma all'interno di un ambiente virtuale. Potrebbe essere utile ad esempio per persone con disabilità motorie, che potrebbero immergersi in questa realtà parallela per fare la spesa, oppure per partecipare ad un concerto.

Attraverso il concetto di NFT (Non Fungible Token)[4], il metaverso potrebbe espandersi anche a campi come la moda o la cultura: si pensi a poter accedere ad una sfilata nel mondo virtuale, o ad un museo.

La peculiarità di questa nuova tecnologia è quella di rappresentare l'autenticità di un prodotto(un'opera d'arte o un pregiato capo di moda) attraverso la blockchain, struttura dati condivisa e immutabile, assicurando quindi all'oggetto di non poter essere replicato e quindi di mantenere la sua autenticità.

Digital Twin e Realtà Virtuale

Esiste una stretta correlazione tra Digital Twin e realtà virtuale. Infatti, un Avatar è la rappresentazione nel mondo virtuale di una persona(o di un oggetto) che agisce in corrispondenza ad una sua controparte fisica, esistente nel mondo reale.

L'Avatar si comporta nell'ambiente virtuale come se fosse una persona, compiendo azioni, immagazzinando informazioni e interagendo con gli altri elementi presenti nel suo ambiente a nome della sua controparte fisica.

Queste tecniche sono state usate prevalentemente da parte di giochi e siti web, ma potrebbero essere usate anche nell'ambito dell'istruzione o in altri

ambiti. Possono essere poi un modo per rendere le interfacce utente più appetibili esteticamente, e anche più fruibili da chi non è avvezzo all'utilizzo dei sistemi elettronici.

Alcune importanti feature dell'architettura software di queste tecnologie applicabili ai Digital Twin sono:

- Uno scambio continuo di dati deve mantenere sincronizzati i Digital Twin alla loro controparte fisica
- Gli oggetti fisici e quelli digitali devono essere sempre allocati nel tempo e nello spazio, questo concetto è molto importante anche per poi realizzare simulazioni e previsioni tramite l'utilizzo di tecnologie di intelligenza artificiale
- Deve esistere un modello che possa rappresentare al meglio l'asset fisico nel mondo virtuale
- Lo spazio virtuale e quello fisico dovrebbero rappresentare gli stessi vincoli

1.2.3 Influenze dai Sistemi Multi-Agente

L'architettura software multi-agente comprende molte similarità con i Digital Twin. I sistemi multi-agente sono basati sull'implementazione di agenti, che agiscono in un certo ambiente collaborando tra loro e raccogliendo dati al fine di raggiungere un determinato obiettivo.

In genere, richiedono cooperazione tra di loro, e quindi somigliano molto, ad esempio, ad una fabbrica composta da tanti macchinari che devono interagire tra loro (un macchinario può essere visto come un digital twin).

I sistemi multi-agente possiedono proprietà interessanti cui i Digital Twins possono beneficiare:

- Un agente rappresenta una entità esterna, e agisce come tale, all'interno di un ambiente specifico.
- Un agente può essere di differenti tipologie: passivo, attivo, o cognitivo. Un agente passivo rappresenta oggetti che hanno un contributo minimo al raggiungimento di un obiettivo all'interno dell'ambiente, e che quindi possono essere una roccia, un oggetto statico, e così via.

  Un agente attivo invece contribuisce al raggiungimento dell'obiettivo comune, mentre quello cognitivo è un agente che è dotato di una intelligenza. Tecniche come il Machine Learning possono aiutare a raggiungere questo status.

- Gli agenti operano in ambienti tipicamente molto complessi e difficili da modellare.
- Gli agenti possono mantenere un grande quantitativo di dati allo scopo di incrementare la propria intelligenza.

1.2.4 Contributi dalla virtualizzazione

La virtualizzazione è l'insieme delle tecniche che consente di creare un ambiente di elaborazione virtuale rispetto ad un ambiente fisico.

Ha avuto un impatto rilevante nello sviluppo di Internet, ed è stato applicato fortemente soprattutto nell'Information Technology e negli ambienti di rete.

Si basa sull'astrazione dall'hardware offrendo la possibilità di utilizzare diversi ambienti di esecuzione sugli stessi calcolatori; viene da se quindi pensare che questa idea sia piuttosto in linea con quella di Digital Twin, che vuole astrarre invece un oggetto fisico.

Anche dal punto di vista informatico, i sistemi di virtualizzazione offrono vari meccanismi utili all'implementazione di Digital Twin, offrendo meccanismi come sicurezza, analisi delle performance e scalabilità.

1.3 Proprietà Caratterizzanti dei Digital Twin nel contesto dell'IoT

Questa sezione vuole presentare la definizione di Digital Twin attraverso le sue proprietà fondanti. Le proprietà definite dal paper vogliono essere abbastanza generali per essere applicate in molti ambiti differenti, ma anche abbastanza specifiche per essere sufficientemente esaustive nel ricoprire le necessità di rappresentazione per quanto riguarda ogni ambito.

1.3.1 Representativeness and Contextualization

La rappresentazione logica di un oggetto dovrebbe essere quanto più verosimile a quella dell'oggetto fisico. Rappresentare tutte le proprietà e le varie sfaccettature però può essere molto complicato, e spesso inutile.

Dovrebbero essere rappresentate le proprietà e le caratteristiche che siano necessarie e sufficienti a qualificare l'oggetto logico come rappresentativo di quello fisico pensando a tutte le sue parti che si vogliono analizzare.

La rappresentatività dovrebbe essere considerata in rapporto a tre importanti parametri:

- Similarità: quanto e come la rappresentazione riproduce l'oggetto fisico.
- Casualità: la probabilità che l'oggetto logico abbia uno stato diverso da quello dell'oggetto fisico.
- Contestualizzazione: è importante considerare queste proprietà all'interno del contesto di riferimento, perchè in base a questo possono cambiare le proprietà che si vogliono rappresentare.

1.3.2 Reflection

Il modello del Digital Twin deve riflettere nel modo corretto l'oggetto fisico, ed è quindi importante capire se sia abbastanza rappresentativo dell'oggetto considerato.

Spesso esistono entità fisiche non facilmente rappresentabili, come ad esempio il corpo umano, e quindi, chi crea il modello deve decidere quali siano le parti più importanti su cui concentrarsi di più.

Ogni valore rilevante dell'oggetto fisico dovrebbe essere univocamente rappresentato nel modello logico. La funzione che rappresenta il mapping dei dati fisici a quelli digitali non deve essere però per forza iniettiva: applicando tecniche di intelligenza artificiale, multiple proprietà dell'oggetto fisico potrebbe essere ricondotte ad una singola nel Digital Twin.

1.3.3 Replication

È l'abilità di replicare un oggetto fisico in diversi contesti, infatti una delle più importanti proprietà di un Digital Twin è quella di poter essere "clonato" e utilizzato in contesti differenti da quello di partenza.

Grazie alla virtualizzazione è possibile virtualizzare i Digital Twin in maniera facile, come si fa con le componenti software. I Digital Twin "clonati" possono poi essere aggiornati direttamente tramite l'oggetto fisico, o tramite una sua replica digitale(Master Replica).

L'utilizzo di una "Master Replica" è utile al fine di evitare multiple richieste di polling, infatti si può collegare **una** rappresentazione virtuale all'oggetto fisico, per poi connettere alla copia digitale più parti virtuali. In questo modo l'asset fisico comunicherà soltanto con una rappresentazione, diminuendo il carico di computazioni a suo carico.

1.3.4 Entanglement

Questa proprietà si riferisce al link presente tra copia digitale e la sua controparte fisica, in termini di scambio di informazioni che devono essere

passate (in breve) tempo verso il Digital Twin. È molto importante che lo scambio di informazioni sia affidabile, perchè la parte digitale necessita di avere una visione aggiornata dell'oggetto fisico allo scopo di mettere a disposizione queste informazioni alle applicazioni che lo utilizzano. L'Entanglement viene definito in tre caratteristiche:

- Connectivity: la comunicazione attraverso le parti è strettamente correlata alla potenza computazionale messa a disposizione dall'oggetto fisico, non è nemmeno sempre possibile che sia in grado di comunicare autonomamente, e in questi casi si possono utilizzare parti terze per effettuarla (es. Un dispositivo che utilizza un broker MQTT). Esiste anche la possibilità di inferire lo stato dell'oggetto fisico tramite l'osservazione diretta (es. Videocamera di sorveglianza).

- Promptness: A seconda del contesto in cui deve avvenire la comunicazione esistono ordini di grandezza temporali differenti per determinare quale sia la latenza minima della comunicazione. In linea di massima si può applicare questo principio: il tempo impiegato per effettuare i cambiamenti di stato deve essere non significativo rispetto a quello impiegato dall'utente per usufruire dell'oggetto. Ad esempio, se avessimo il Digital Twin di un parcheggio, bisognerebbe che la comunicazione fosse effettuata in meno tempo rispetto a quello impiegato da una macchina per entrarne ed uscirne.

  Gli ordini di grandezza possono variare moltissimo: si pensi ad una applicazione che lavora in campo medico, e che quindi ha bisogno della massima reattività, e ad un sistema che rappresenta i posti liberi di una biblioteca da 1000 posti: in questo caso la reattività degli aggiornamenti è trascurabile.

- Association: La comunicazione tra Digital Twin e oggetto fisico può essere unidirezionale, o bidirezionale. Tipicamente, la direzione è quella dall'oggetto fisico alla copia digitale, ma anche una comunicazione bidirezionale può avere molto valore: ad esempio in un sistema composto da attuatori.

1.3.5 Persistency

Il Digital Twin deve essere persistente nel tempo, ed essere sempre disponibile, anche quando la sua controparte fisica non lo è. In caso di malfunzionamenti inoltre, la copia digitale dovrebbe interessarsi di re-stabilire un collegamento con l'oggetto fisico e di re-sincronizzarsi.

1.3.6 Memorization

È importante che i Digital Twin mantengano la loro storia, intesa come storico dei loro dati. È infatti fondamentale per usare tecniche riguardanti l'intelligenza artificiale, e quindi per poter effettuare analisi sui comportamenti dell'oggetto, avere a disposizione una grande mole di dati da poter analizzare.

La quantità di dati dovrebbe essere *il più possibile*, ma è fondamentale che i dati siano di qualità, altrimenti la quantità risulta inutile e anche dannosa (predizioni errate).

1.3.7 Composability

Nella vita reale, gli oggetti sono spesso composizioni di varie sottoparti. Ad esempio, un'auto è composta dalle sue ruote, dal motore, dalla carrozzeria, e così via. Un Digital Twin dovrebbe replicare questa proprietà, in modo che ogni parte sia allo stesso tempo isolata e connessa alle altre.

A seconda del contesto applicativo il creatore del modello può determinare la granularità con cui effettuare questa "divisione", anche a seconda dell'oggetto in questione.

Un compito importante è poi lasciato anche ai creatori dei framework e delle piattaforme che supportano i Digital Twin: creare un modello che possa rappresentare la modularità degli oggetti senza perdere la visione d'insieme necessaria sull'oggetto.

1.3.8 Accountability/Manageability

Gestire in maniera precisa ed accurata i Digital Twin è una proprietà molto importante, essi dovrebbero essere sempre disponibili e anche quando l'oggetto fisico va offline, dovrebbero mettere a disposizione le informazioni più recenti in loro possesso e avviare un processo per ristabilire la connessione con l'oggetto. Inoltre, visto che il Digital Twin può venire utilizzato da varie applicazioni, la piattaforma deve essere scalabile ed efficiente per permettere di essere usato in maniera fruibile.

1.3.9 Augmentation

Un oggetto fisico ha dei servizi e delle funzionalità ben definite, che sono sempre le stesse per il suo intero ciclo di vita. Questo non vale per la sua controparte digitale, che invece, sulla base dei dati ricevuti dall'oggetto fisico, può creare funzioni nuove che vanno ad aumentare le sue funzionalità. Anche

un oggetto statico, come può essere una statua, può essere reso più "intelligente" aggiungendo funzioni per recuperare delle sue caratteristiche (materiale, autore, data di pubblicazione).

1.3.10 Ownership

Come detto in precedenza, i Digital Twin tengono al loro interno una importante mole di dati, ed è quindi importante gestire chi può accedere, e chi può usarli.

Inoltre, le rappresentazioni logiche e quelle fisiche possono non condividere gli stessi diritti d'uso e di lettura: un quadro è proprietà del pittore che lo crea, ma ad esempio la sua copia digitale potrebbe essere data in licenza a più musei che lo espongono. Anche la proprietà di Augmentation ha un ruolo: se vengono aggiunte funzionalità, è giusto che il proprietario del Digital Twin con quelle funzionalità sia di chi le ha create.

1.3.11 Servitization

Si riferisce alla possibilità di offrire sul mercato un insieme di servizi software legati ad un oggetto fisico. È strettamente legato alla proprietà di Augmentation, in quanto si basa sull'aggiunta di funzionalità rispetto ad un oggetto fisico "base". Le aziende potrebbero quindi vendere prodotti che non sono più soltanto oggetti ma insieme di funzionalità. Anche il rapporto tra aziende/-clienti potrebbe essere migliorato, sempre mantenendo comunque i giusti ruoli attraverso la proprietà di Ownership.

Nel lungo termine, la Servitization potrebbe creare un nuovo tipo di economia basata non più sul possedimento degli oggetti fisici, ma invece sul "pay per usage".

1.3.12 Predictability

I Digital Twin rappresentano una grande mole di proprietà ed eventi, che provengono dalla loro controparte fisica. È capace di operare in contesti ben noti e di interagire con altri oggetti.

La proprietà di Predictability si riferisce alla possibilità di usare il Digital Twin in diversi ambienti (contesti) al fine di eseguire simulazioni sul suo comportamento e sulle interazioni con altri oggetti, nel futuro o in un periodo specifico di tempo. Queste simulazioni possono essere svolte tramite l'analisi delle sue proprietà, utilizzando le recenti tecniche che l'intelligenza artificiale mette a disposizione.

Inoltre, utilizzando sistemi IoT, i Digital Twin potrebbero essere utili anche in contesti molto grandi e complessi come città, fabbriche e reti logistiche.

1.4 Panoramica sulle piattaforme e framework per Digital Twin

Il paper fin qui presentato non si esprime sulla parte implementativa dei Digital Twin, ma ne esprime soltanto i concetti fondamentali.

In questa sezione illustrerò alcune soluzioni che negli anni si sono sviluppate all'interno del mercato, dopo aver fatto questa panoramica, mi soffermerò su Eclipse Ditto, la piattaforma che ho approfondito e che ho utilizzato per sviluppare una mia applicazione basata su Digital Twin.

1.4.1 Panoramica delle soluzioni

- General Electric: L'azienda è stata tra le prime a sviluppare soluzioni per l'IoT visto in ottica industriale, offrendo una vasta gamma di servizi. È proprio l'ambito industriale quello su cui si focalizzano maggiormente, offrendo applicazioni come SmartSignal (prevenzione malfunzionamenti) e Asset Performance Management (insieme di servizi per ottimizzare le prestazioni).

- Azure Digital Twins: è un servizio che risiede sul cloud, di tipo "Platform as a Service", che permette di creare modelli digitali anche di ambienti complessi come stadi, ospedali, o addirittura intere città.

- Siemens: è una delle aziende che stanno ponendo le basi per l'industria 4.0. Si focalizza infatti sui prodotti, come all'inizio della storia dei Digital Twin sosteneva Grieves. Esistono software per ottimizzare sia la fase di produzione che quella di utilizzo (ottimizzando le performance di un prodotto).

- IBM: i Digital Twin vengono utilizzati per il testing, il monitoraggio e la creazione di prodotti e processi attraverso software come IBM Engineering Lifecycle Management e IBM Engineering Systems Design Rhapsody.

- Cisco Systems: l'infrastruttura industriale di Cisco sfrutta i Digital Twin per creare soluzioni affidabili, intelligenti e sicure. Attraverso il Cisco DNA Center è possibile configurare i device sulla rete in poco tempo, e attraverso l'uso di software come Cisco Kinetic IoT platform è possibile raccogliere e mantenere i dati provenienti dai device.

- Eclipse Ditto: offre un middleware per la creazione, l'utilizzo e la gestione di Digital Twin. Verrà approfondito nel prossimo capitolo.

Capitolo 2

Eclipse Ditto

2.1 Introduzione

Eclipse Ditto è un framework open source, utile per l'implementazione dei Digital Twin. Il suo scopo è quello di creare una rappresentazione virtuale di oggetti esistenti nella realtà, come possono essere macchine intelligenti, stazioni di ricarica, ecc. Queste rappresentazioni si chiamano *Twins*, e rendono possibile per l'utente accedere all'oggetto fisico come se fosse un qualsiasi servizio web. Viene inoltre offerta la possibilità di sfruttare protocolli di comunicazione già esistenti per comunicare da/verso i device.

Eclipse Ditto non vuole essere una piattaforma IoT (Internet of Things) completa, infatti nessuna sua parte esegue su hardware IoT e non è definito nessun protocollo per la comunicazione verso i device.

2.2 Quando usarlo

Una soluzione IoT che riguarda una componente hardware ed una software è una situazione in cui Ditto può risultare utile. Viene offerta una API, completa di parti per l'autenticazione, che è disponibile su HTTP e WebSocket. Il suo scopo è quello di interagire con i Digital Twin, in modo da poterli creare, gestire ed utilizzare. Tra le sue funzioni principali ci sono:

- Astrarre le componenti hardware.
- Mandare messaggi da/verso hardware e applicazioni.
- Garantire accessi autorizzati.
- Avere una memoria persistente e offrire i dati dell'hardware anche quando non è connesso.

- Mantenere traccia dei cambiamenti notificando le parti interessate.

2.3 Digital Twin secondo Ditto

Uno dei problemi più grandi con i Digital Twin è la loro definizione: ne esistono molte e non ne esiste una univoca.

Secondo Ditto, un Digital Twin sarebbe un concetto per astrarre un oggetto del mondo reale nella sua rappresentazione digitale, comprendente tutte le sue capacità ed aspetti, che può essere applicato sia in ambito industriale che in ambito IoT. Un framework per Digital Twin dovrebbe permettere di interagire con molti twins, e assicurare che gli accessi vengano eseguito solo dalle parti autorizzate.

2.4 Architettura

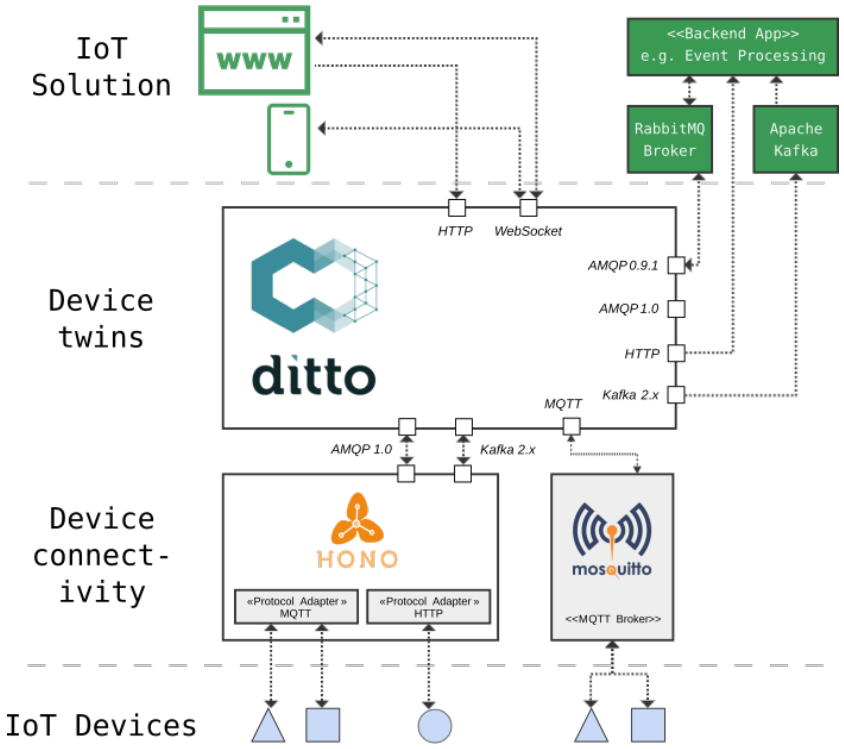

Figura 2.1: Schema Architetturale di Ditto

Ditto agisce come un middleware IoT (figura 2.1), effettuando una astrazione degli oggetti fisici nelle soluzioni IoT attraverso i Digital Twin. C'è la

possibilità di interfacciarsi ai device con diversi tipi di protocolli attraverso l'uso di connessioni, dando possibilità alle soluzioni IoT di interagire con gli oggetti fisici. La connessione di Ditto deve essere anche indirizzata verso un layer di connettività, che fa da tramite da/verso il device, come Eclipse Hono ed Eclipse Mosquitto. Per quanto riguarda invece la connettività verso le soluzioni IoT Ditto offre connessioni HTTP e WebSocket.

2.5 Entità del Modello

Il modello di Ditto è composto principalmente dalle seguenti entità:

- Thing
- Feature
- Policy

2.5.1 Thing

I Thing sono le entità che rappresentano l'oggetto fisico come Digital Twin all'interno di Ditto. Alcuni esempi possono essere:

- Entità Fisiche: un sensore, un veicolo, una lampada.
- Entità Virtuali: meteo di una località, una stanza.
- Entità Transazionali: un viaggio, un'insieme di misurazioni su una macchina.

Sono provvisti di un identificativo, chiamato *ThingId*, che deve sottostare a delle regole ben precise riguardanti i caratteri permessi e quelli vietati. Questo a causa dell'uso nelle richieste HTTP. L'accesso ad un Thing è regolato da una *Policy*, che gestisce le parti che possono avere permessi di *READ* e di *WRITE* sul Thing. Può inoltre contenere una definizione, in modo da specificare meglio le sue proprietà e comportamenti. È possibile inserire definizioni provenienti da Eclipse Vorto e dal Web Of Things.

Le proprietà vengono modellate tramite:

- Attributes
- Features

Attributes

Gli attributi descrivono il Thing, e possono essere anche usati per identificarli. Sono usati per modellare le proprietà più statiche. Esempi possono essere il numero seriale e la data di fabbricazione di una macchina.

Features

Le Feature hanno un identificativo come i Thing, e allo stesso modo, sono sottoposti a regole ben precise riguardanti i caratteri permessi e quelli vietati. Sono usate per modellare le parti dinamiche, e ognuna contiene una lista di proprietà. Le proprietà possono essere categorizzate, e l'insieme di esse rappresenta un oggetto JSON; ognuna di esse può essere un valore semplice, o un oggetto complesso, sono permessi tutti i tipi supportati da JSON.

Esiste la possibilità di specificare le *Desired Properties*, che rappresentano uno stato delle proprietà. Possono essere usate quando si è interessati al raggiungimento da parte di una proprietà di un certo stato.

Ditto non fornisce per ora nessuna implementazione a riguardo, che quindi spetta al programmatore.

Come i Thing, le Feature possono possedere una definizione che ne specifica meglio capacità e comportamenti. Anche qui gli standard sono Eclipse Vorto e Web Of Things.

Non viene assicurato però che i tipi delle proprietà seguano quelli forniti dalla definizione.

```
{
  "thingId": "the.namespace:thing01",
  "policyId": "the.namespace:policy01",
  "attributes": {
    "location": "Kitchen"
  },
  "features": {
    "transmission": {
      "properties": {
        "cur_speed": 90
      }
    }
  }
}
```

Listato 2.1: Modello di un Thing

Nel modello qui sopra riportato, si possono notare:

- un attributo *location* con valore "Kitchen".

- una feature *transmission* con la proprietà "cur_speed".

2.5.2 Policy

Le Policy vengono utilizzate all'interno di Ditto per regolare l'accesso ai Thing e alle altre entità, a diversi livelli di granularità. Più precisamente, esse concedono l'accesso ad un *Subject*, specificando un accesso di READ/WRITE per una certa risorsa.

Subjects

I Subject sono coloro che possono avere l'accesso ad una certa risorsa, e Ditto offre diversi modi per autenticarsi come tali. In particolare:

- Nginx (Pre-Authentication)
- Google (JSON Web Token)

Sfruttando Pre-Authentication e JSON Web Token, Ditto offre anche la possibilità di autenticarsi con provider diversi da Nginx e Google, la soluzione più semplice per chi si approccia a Ditto è però Nginx, dato che viene integrato nella installazione.

Expiring Subjects

Esiste la possibilità di specificare un timestamp nel Subject di una Policy. In questo modo, l'accesso ad una determinata risorsa è garantito solo per un determinato periodo di tempo. Inoltre, è possibile anche configurare le politiche d'arrotondamento, in modo che sia le situazioni in cui una risorsa è particolarmente sensibile, e sia quelle in cui non lo è, vengano gestite al meglio.

Resources

Le Resources sono le parti a cui possono o non possono accedere i Subject. Possono essere:

- Policy
- Thing
- Feature
- Message

```
{
   "entries":{
      "owner":{
         "subjects":{
            "nginx:ditto":{
               "type":"nginx basic auth user"
            }
         },
         "resources":{
            "thing:/":{
               "grant":[
                  "READ",
                  "WRITE"
               ],
               "revoke":[

               ]
            },
            "policy:/":{
               "grant":[
                  "READ",
                  "WRITE"
               ],
               "revoke":[

               ]
            },
            "message:/":{
               "grant":[
                  "READ",
                  "WRITE"
               ],
               "revoke":[

               ]
            }
         }
      }
   }
}
```

Listato 2.2: Il modello di una semplice Policy

Nel modello qui sopra riportato sono stati concessi i permessi READ e WRITE a tutte le risorse.

2.6 Messages

I Messages offrono la possibilità di comunicare da/verso un device fisico. Secondo Ditto i messaggi dovrebbero essere rivolti:

- **Verso** i dispositivi quando si tratta di invocare un'azione (i.e spegni/accendi)
- **Da** i dispositivi quando si tratta di notificare un evento/allarme (i.e malfunzionamenti).

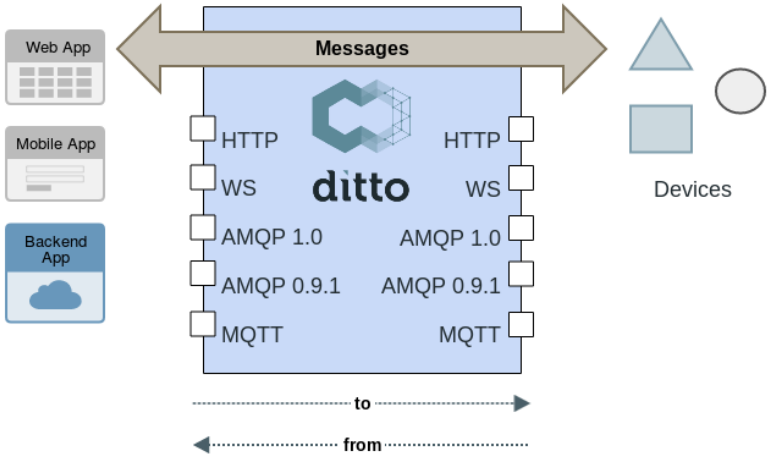

Figura 2.2: Come Ditto instrada i messaggi

Come si vede dalla figura 2.2, Ditto non svolge alcun ruolo da "mediatore", e il suo ruolo riguarda solamente l'instradamento verso il destinatario. Non è presente una parte sul QoS (Quality of Service) dei messaggi, e Ditto assicura soltanto che i messaggi siano instradati *at most once*. Inoltre, se un messaggio viene mandato quando la parte deputata a riceverlo è offline, non verrà mandato nuovamente quando la parte si riconnetterà.

2.6.1 Elementi

I messaggi sono formati da vari elementi obbligatori:

- **Direzione** del messaggio, a seconda che sia mandato o ricevuto da un Thing (Inbox o Outbox).
- **ThingId** del Thing che riceve/manda il messaggio.
- **Subject/Topic** del messaggio.

Esistono poi altre parti facoltative:

- **FeatureId** della Feature che riceve/manda il messaggio.
- **Content-Type** se si vogliono mandare dati nel messaggio.
- **Correlation-Id** che permette di instradare l'eventuale risposta ad un messaggio.

2.6.2 APIs

I messaggi possono essere mandati attraverso:

- WebSocket API in Ditto Protocol.
- HTTP API, sia in maniera bloccante(per ricevere una risposta) che non.
- Tramite una Connection Source in Ditto Protocol.

Possono essere ricevuti invece tramite:

- WebSocket API in Ditto Protocol.
- ServerSentEvents(SSE) API.
- Tramite un Connection Target.

Per ricevere e mandare i messaggi, sono richiesti rispettivamente i permessi di *READ* e *WRITE* sulla risorsa *message*. Inoltre, c'è bisogno di possedere i permessi anche per quanto riguarda la risorsa *thing*.

2.7 Signals

Il concetto di "Signals" viene usato principalmente per gestire le comunicazioni interne nel cluster di Ditto, ma è comunque importante per l'utente capirne il significato e scoprire in quali pattern di comunicazione vengono usati.

Essi si dividono in:

- Commands

- Command Responses
- Error Responses
- Events
- Announcements

2.7.1 Commands

Vengono usati per interagire col dominio e il loro nome è un verbo coniugato all'imperativo. Essi rappresentano una richiesta, che quindi può essere anche rifiutata. A seconda di come interagiscono col dominio possono essere *Modify Command* o *Query Commands*. I primi, vengono usati per modificare una parte di un Twin , ma anche per creare o eliminare. I secondi invece, vengono utilizzati solamente per recuperare dati.

2.7.2 Command Responses

Sono la risposta ai Commands, e rendono noto se essi hanno effettivamente modificato il dominio. Nel caso dei *Query Commands* invece vengono riportati i dati richiesti. In entrambi i casi, nel caso di errori viene ritornato una *Error Response*.

2.7.3 Error Responses

Nel caso in cui una modifica non possa essere applicata, o una informazione non possa essere recuperata, viene ritornato un errore.

2.7.4 Events

Rappresentano il fatto che sia avvenuto qualcosa all'interno del dominio. È importante sottolineare che un evento riguarda un qualcosa che è **già** successo, e che quindi non può più essere annullato o modificato.

Essi sono:

- mantenuti/aggiunti all'interno dell'archivio dati interno di Ditto.
- pubblicati all'interno del cluster Ditto, in questo modo gli altri servizi possono reagire.
- sono pubblicati attraverso le varie API alle parti interessate ed autorizzate.

2.7.5 Announcements

Sono pubblicati appena prima che qualcosa succeda all'interno del dominio, e al contrario degli *Events*, essi annunciano che un qualcosa succederà a **breve**. Hanno le seguenti caratteristiche:

- non sono mantenuti all'interno dell'archivio dati interno di Ditto.
- sono pubblicati attraverso le varie API alle parti interessate ed autorizzate.

Terminata la discussione dei vari tipi di *Signals*, possiamo esemplificare il pattern di comunicazione:

1. un *Command* viene mandato a Ditto, che lo processa
2. viene ritornata una *Success Response* o una *Error Response* a chi ha mandato il *Command*.
3. un *Event* viene aggiunto all'archivio dati, e può essere ricevuto dalle parti interessate.

2.8 APIs

Ditto offre due API:

- una HTTP API REST-like.
- una WebSocket API.

Le due opzioni offrono praticamente le stesse possibilità di interazione con i Thing, ma allo stesso tempo hanno differenze importanti.

2.8.1 HTTP API

La HTTP API è un protocollo di tipo *Connectionless* e quindi ha un consumo di risorse più limitato, dato che la connessione viene aperta solo quando arrivano le richieste. La sicurezza del canale è assicurata dal protocollo HTTPS(HTTP over Transport Layer Security) e le richieste fatte alla API sono bloccanti: essa lavora in modo sincrono. L'autenticazione può essere gestita tramite:

- HTTP BASIC Authentication
- JSON Web Token (JWT) issued OpenID connect provider

Attualmente la versione 1 è stata deprecata e la versione più aggiornata è la 2. Attraverso la API si possono modificare Things, Features, Policies, oltre a poter mandare Messaggi e cercare Things.

Server Sent Events

Una parte importante della API riguarda i SSE[5](Server Sent Events).

Essi possono essere usati per essere notificati riguardo i cambiamenti di stato di un Thing, oppure per ricevere Messaggi.

Infatti, essi stabiliscono una connessione unidirezionale dal back-end verso il client, non consentendogli alcuna comunicazione in uscita, ma solo di mettersi in ascolto.

2.8.2 WebSocket

La WebSocket è un protocollo *Connection-oriented* che quindi mantiene una connessione sempre aperta(stabilita con un solo handshake) per ottimizzare la latenza e il throughput.

Il protocollo di sicurezza è il WSS(WebSocket over Transport Layer Security) e le richieste fatte alla WebSocket sono asincrone.

I protocolli per l'autenticazione rimangono i medesimi della HTTP API.

Si basa fortemente sul Ditto Protocol, che stabilisce il tipo di messaggi mandati attraverso il protocollo.

Il Ditto Protocol sarà approfondito maggiormente nella sottosezione dedicata.

Vantaggi rispetto la HTTP API

1. Una volta che la connessione è stata aperta, non esistono header, e non c'è più bisogno di stabilire connessioni. Attraverso la WebSocket si possono scambiare più messaggi nello stesso periodo di tempo.

2. Si tratta di una connessione duplex, e può quindi mandare e ricevere dati in entrambe le direzioni. Alcune feature di Ditto come le *Change Notification* sono quindi ben supportate, ma è interessante notare che anche la HTTP API tramite i SSE può farne uso.

3. Le varie comunicazioni possono essere scambiate anche tramite multiple sessioni WebSocket, ma bisogna tenere a mente che Ditto non offre nessun tipo di scheduling per notificare le varie sessioni.

2.9 Ditto Protocol

Il Ditto Protocol definisce un protocollo per il formato dei messaggi basato su JSON utilizzato per comunicare con i Twin e i Device. Esso consente di comunicare attraverso due canali: Twin e Live.

2.9.1 Canale Twin

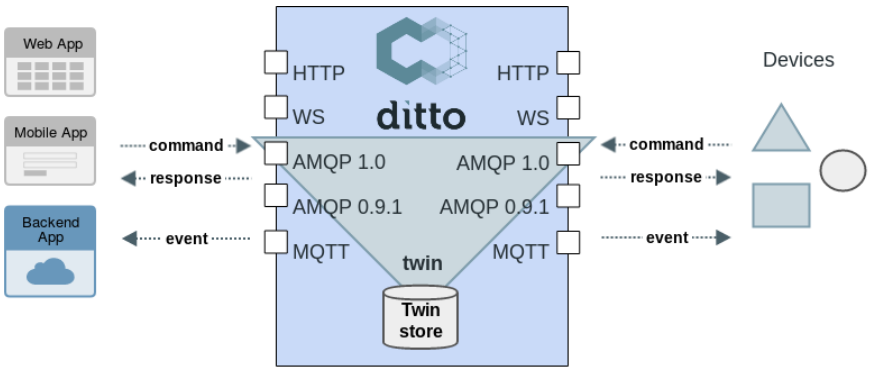

Figura 2.3: Canale Twin di Ditto

La figura 2.3 è molto chiara: il canale connette i Twin ai Device, ed i Twin alle App che li utilizzano.

Si nota poi che la direzione delle frecce ricalca le definizioni dei *Signals* vista in qualche sezione fa: i Device mandano i Command, cui è possibile ricevere risposta, e sono aggiornati tramite gli Event(conseguenza dei Command).

La situazione è analoga per quanto riguarda il lato Twin-App.

2.9.2 Canale Live

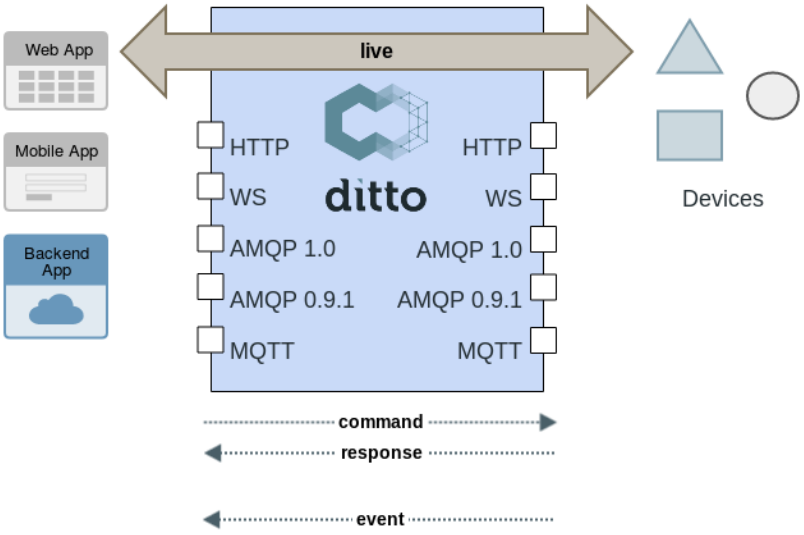

Figura 2.4: Canale Live di Ditto

Nella Figura 2.4 si nota che il canale Live connette i device direttamente alle App che li utilizzano. Ditto in questo caso fa solo un lavoro di instradamento, assicurando però che le parti coinvolte siano autorizzate.

Esempi in cui questo canale risulta utile sono azioni (qualcosa deve avvenire sul device) ed eventi (qualcosa è successo nel device).

2.9.3 Specifiche

Per essere un messaggio valido esso deve comprendere:

- Ditto Protocol Envelope
- Ditto Protocol Payload

La *Envelope* è definita precisamente dal protocollo di comunicazione su cui viaggiano i dati, ma deve comunque sottostare alle regole di uno schema JSON. All'interno del campo *value* è contenuto il payload del messaggio, che contiene la richiesta vera e propria, riguardante ad esempio Things o Policies.

2.10 Connections

Le connessioni vengono usate per connettere Ditto a servizi esterni come Eclipse Mosquitto, Eclipse Hono, Apache Kafka. Un esempio importante è quello della connessione da/verso un device, dato che Ditto non offre soluzioni specifiche a riguardo, oppure quello di inviare dati su un database esterno.

Attualmente, vengono offerti i seguenti tipi di connessione:

- AMQP 0.9.1
- AMQP 1.0
- MQTT 3.1.1
- MQTT 5
- HTTP 1.1
- Kafka 2.x

Le connessioni puntano ad essere più generiche possibili rispetto al tipo usato, mantenendo una struttura comune per tutte. Nonostante ciò, è possibile utilizzare le personalizzazioni offerte da ogni connessione.

Tratterò quindi di seguito le componenti comuni, senza soffermarmi sulle specifiche configurazioni di ogni tipo.

2.10.1 Source

Sono utilizzate per ricevere i messaggi che provengono da servizi esterni. Contengono tipicamente gli indirizzi utilizzati per la ricezione dei messaggi da parte dei servizi esterni ed un subject di autorizzazione. Inoltre, possono essere configurate funzionalità come acknowledgement, filtri, o header mapping.

2.10.2 Target

Sono utilizzati per pubblicare messaggi verso servizi esterni. Ci sono le stesse parti presenti nella Source, ma con l'aggiunta di un *Topic*. Esso rappresenta quali parti del dominio Ditto si vogliono mandare, come possono essere messaggi ed eventi. In questo modo si può continuare ad utilizzare Ditto per gestire il Digital Twin avendo in automatico le informazioni reindirizzate verso il device, oppure verso servizi esterni che possono gestire come vogliono le informazioni.

2.10.3 Payload Mapping

Ditto necessita di ricevere messaggi in DittoProtocol per eseguire modifiche riguardanti il modello, ma non sempre i device hanno abbastanza potenza computazionale per poterlo fare: spesso i dati sono in formato JSON semplice, oppure in alcuni casi, addirittura in formati più a basso livello. C'è quindi la necessità di poter mappare il payload dei messaggi, e Ditto offre diversi tipi di mapper, sia per mappare quello in entrata, che quello in uscita.

Secondo i miei usi, quello che è risultato più utile è il mapper in Javascript, che permette di transformare messaggi in qualsiasi formato in DittoProtocol, escludendo quindi la piattaforma Ditto dalla gestione di formati sconosciuti.

Capitolo 3

Web of Things e Integrazione con Eclipse Ditto

Eclipse Ditto, nella versione 2.4.0, ha aggiunto una integrazione per il Web Of Things, progetto del W3C che si pone come obiettivo quello di creare uno standard per la modellazione di dispositivi IoT. Tra i casi d'uso citati, sono presenti anche i Digital Twin[6].

3.1 Web Of Things

3.1.1 Introduzione

Gli sviluppatori che lavorano ai progetti IoT si trovano di fronte a una situazione difficile. L'insieme di tecnologie che compone un sistema IoT può comprendere numerosi sistemi e servizi provenienti da differenti fornitori. Queste differenze si articolano in differenti protocolli di comunicazione, differenti modelli per lo scambio di dati e anche in requisiti di sicurezza diversi. Le applicazioni IoT vengono quindi sviluppate con grandi sforzi, ma ottenendo soluzioni specifiche per il caso d'uso utilizzato, e questo rende le soluzioni difficili da estendere, mantenere o riusare.

Il Web Of Things si propone di creare un insieme di tecnologie standardizzate, che possano semplificare lo sviluppo delle applicazioni IoT, utilizzando come esempio gli standard Web, che si sono rivelati negli anni a venire, di successo. L'approccio permette di aumentare la **flessibilità** e **interoperabilità**, sbloccando l'enorme potenziale commerciale racchiuso nell'IoT, ora bloccato dalla frammentazione.

3.1.2 Building Blocks

Il W3C struttura il Web Of Things come composizione di *Building Blocks*, parti che hanno la loro valenza da sole, ma che sono interoperabili. Segue una loro panoramica:

- Thing Description: descrive i metadati e le interfacce dei Things, intesi come astrazioni di entità fisiche o virtuali che possiedono interazioni e partecipano al Web Of Things. Possono essere contenute all'interno del device, oppure da un altro servizio nel caso in cui ci siano problematiche di spazio o potenza computazionale.

- Binding Templates: forniscono le linee guida per definire le interfacce di rete nei Things. Sono utili a risolvere il problema degli svariati protocolli di comunicazione utilizzati dalle soluzione IoT.

  Risolvono questo problema definendo dei vocabolari (Binding Template) che descrivono le comunicazioni tra Things e le integrano all'interno delle Thing Description.

- Scripting API: fornisce le definizioni per creare API atte ad "utilizzare" il Thing, in maniera simile a delle API per un servizio Web.

- Security and Privacy Guidelines: fornisce le linee guida per una implementazione sicura dei Things.

3.1.3 Thing Description

Come detto in precedenza, una Thing Description esprime il concetto di Thing, astrazione di una entità fisica o virtuale. Si tratta di una collezione di metadati semantici, che lo descrivono nelle sue parti. Inoltre, possiede delle annotazioni semantiche, che gli permettono di rendere il suo modello più comprensibile. Utilizza il formato JSON-LD.

Formato JSON-LD

Il formato JSON-LD[7] è una estensione del formato JSON, ormai standard per molteplici ambiti, che supporta l'uso di dati sotto forma di link.

Vengono introdotte anche delle nuove parole chiave, che possono essere usate per definire meglio parti del modello, come:

- @type

- @id

- @context

```
{
  "@context": "https://json-ld.org/contexts/person.jsonld",
  "@id": "http://dbpedia.org/resource/John_Lennon",
  "name": "John Lennon",
  "born": "1940-10-09",
  "spouse": "http://dbpedia.org/resource/Cynthia_Lennon"
}
```

Listato 3.1: Esempio in JSON-LD

Come si vede dal codice d'esempio 3.1, JSON-LD è un formato che permette di creare una rete di modelli, standard-based e machine-readable, basati sulle informazioni che il Web mette a disposizione.

Interaction patterns

Sono i pattern tramite i quali un client interagisce una Thing Description:

- Properties: insieme di proprietà che il Thing possiede.
- Actions: insieme di azioni che il Thing può svolgere.
- Events: insieme di eventi che il Thing può emettere.

Tutte e tre gli "Interaction Pattern" vengono descritti all'interno del modello, specificando anche i protocolli e gli indirizzi per accedervi.

```
{
    "@context": "https://www.w3.org/2022/wot/td/v1.1",
    "id": "urn:dev:ops:32473-WoTLamp-1234",
    "title": "MyLampThing",
    "securityDefinitions": {
        "basic_sc": {"scheme": "basic", "in": "header"}
    },
    "security": "basic_sc",
    "properties": {
        "status": {
            "type": "string",
            "forms": [{"href": "https://mylamp.example.com/status"}]
        }
    },
    "actions": {
```

```
        "toggle": {
            "forms": [{"href": "https://mylamp.example.com/toggle"}]
        }
    },
    "events":{
        "overheating":{
            "data": {"type": "string"},
            "forms": [{
                "href": "https://mylamp.example.com/oh",
                "subprotocol": "longpoll"
            }]
        }
    }
}
```

Listato 3.2: Esempio di Thing Description

L'esempio 3.2 rappresenta la Thing Description di una lampada, specificandone i protocolli di sicurezza e i punti d'accesso per le sue proprietà, azioni ed eventi.

3.1.4 Scripting API

Fornisce le definizioni delle Thing Description, consentendo a progetti come Eclipse Thingweb[8] di implementare delle API in grado di leggere una Thing Description e di fornire i suoi "Interaction Pattern" attraverso una applicazione Web. Permette di aumentare la produttività di un sistema IoT, e di aumentarne la portabilità.

3.2 Integrazione con Eclipse Ditto

L'integrazione si basa sul Thing Description Working Draft dell'11 marzo 2022 che non è ancora stata pubblicata come *W3C Recommendation*, e quindi Ditto considera ancora l'integrazione come sperimentale. Gli aspetti dell'implementazione quindi potrebbero essere soggetti a cambiamenti senza la garanzia di retrocompatibilità.

La Working Draft introduce all'interno del Web Of Things i Thing Models, che possono essere usati come template per la generazione di Thing Descriptions. Essi non posseggono alcune parti, come i form e i protocol bindings, che quindi sono destinate ad essere generate dalla piattaforma deputata alla creazione.

I Thing Model si prestano bene a rappresentare i Twins di Ditto, e in particolare, aggiungono anche dei benefici:

- Modellare meglio i dati, specificandone il tipo, valori massimi/minimi e di default, e le unità di misura.
- Fare uso del contesto semantico (JSON-LD) usando ontologie come SAREF[9] e OM-2[10].
- Assicurare Interoperabilità.
- Essere supportati da uno standard importante, creato dal W3C, che è già un leader nel settore degli standard Web.

3.2.1 Mapping dei concetti Web Of Things su Ditto

Mappare i concetti del Web Of Things su Ditto può essere fatto a vari livelli di complessità, quello che la documentazione consiglia e considera come "più avanzato" consiste in una Thing Description che ne comprende altre al suo interno, rappresentanti le features.

Thing Description e Thing Model

Nell'ottica di Ditto, un Thing Model rappresenta un template per la creazione di una Thing Description, e visti in ottica OOP, potrebbero essere pensati rispettivamente come classe e la sua istanza.

Thing Model rappresentante un Thing

Quando un Thing Model rappresenta un Thing, le sue parti sono mappate nel seguente modo all'interno di Ditto:

Elemento WoT	**Elemento Ditto**
Thing	Thing
Properties	Thing Attributes
Actions	Messages to a Thing
Events	Messages from a Thing
Composition via submodel	Thing Features

Thing Model rapprentante una Feature

Elemento WoT	Elemento Ditto
Thing	Feature
Properties	Feature Attributes
Actions	Messages to a Feature
Events	Messages from a Feature

3.2.2 Creare Un Thing sfruttando l'integrazione

Allo scopo di creare un Thing specificando un modello Web Of Things attraverso Ditto bisogna assegnare un Thing Model valido come *definition*.

Il modello deve essere pubblicamente accessibile tramite il suo URL, e allo stato attuale della scrittura di questa tesi, deve essere privo di redirect. A tale scopo è comodo usare Github Gist, o una repository.

Ditto poi scaricherà la definizione, e genererà la Thing Description, poi recuperabile attraverso la HTTP API specificando di accettare il tipo di dato *application/td+json* tramite header.

Seguono i comandi per generare un Thing sfruttando il WoT e per recuperarlo: entrambi utilizzano la HTTP API.

```
curl --location --request PUT -u ditto:ditto
'http://localhost:8080/api/2/things/com.project.thesis:greenhouse01'
    \
--header 'Content-Type: application/json' \
--data-raw '{
    "definition": "thingModelUrl"
}'
```

Listato 3.3: Creazione di un Thing WoT

```
curl -i --location --request GET -u ditto:ditto
 'http://localhost:8080/api/2/things/com.project.thesis:greenhouse01'
     \
--header 'Accept: application/td+json'
```

Listato 3.4: Recupero di un Thing WoT

Capitolo 4

Sperimentazioni: Prototipazione del Digital Twin di una Serra

Allo scopo di approfondire e toccare con mano i concetti elencati in precedenza, ho implementato il Digital Twin di una serra.

4.1 Specifiche

Il sistema che ho voluto realizzare vuole replicare una serra e si basa su tre parti fondamentali:

- Device IoT: il suo compito è quello di rilevare attraverso i sensori ad esso collegati temperatura, umidità e luminosità. Inoltre, possiede un led che vuole simulare una luce artificiale.

- Digital Twin attraverso Ditto: lo scopo è quello di astrarre il device come un servizio web, per poterlo poi utilizzare.

- Applicazione Web: fornisce la visualizzazione dei dati e la possibilità di interagire con il Device.

4.2 Device IoT

Il Device IoT si basa sul microcontrollore ESP32[11], che, collegato ad un insieme di sensori, si occupa di simulare una serra, rilevando le condizioni ambientali presenti all'interno. Di seguito presenterò i sensori utilizzati.

4.2.1 DHT11

Il DHT11 è un sensore a basso costo che offre la misurazione di temperatura ed umidità. Possiede un termistore ed un sensore capacitivo per l'umidità, correlati ad chip ADC (Analog to Digital) che permette di utilizzarlo tramite i pin digitali della board ESP32.

Presenta le seguenti caratteristiche:

- Prezzo molto basso (Al 10 maggio 2022, meno di un euro).
- Funzionamento a 3 e 5 Volt.
- Massimo uso di corrente di 2.5 mA.
- Lettura dell'umidità dal 20% all'80% con una accuratezza del +-5%.
- Lettura della temperatura da 0°C a 50°C con una accuratezza del +- 2°C.
- Sampling rate massimo di 1Hz (Una misurazione al secondo).

4.2.2 Fotoresistore

Il fotoresistore è una resistenza che varia il suo valore in base al livello di luminosità che rileva, mentre un luxmetro è un dispositivo in grado di misurare il livello di Lux che lo irradiano.

Il fotoresistore però, come detto in precedenza, varia solo il suo valore di resistenza, e quindi deve essere calibrato con un luxmetro per fornire una misura dei lux. È possibile quindi, una volta calibrato, utilizzare il fotoresistore come se fosse un luxmetro.

L'idea è quella di costruire una funzione che leghi la misura dei lux misurata empiricamente con un Luxmetro, a quella della resistenza del fotoresistore. Una volta costruita la funzione, è possibile associare la resistenza rilevata ai lux.

È stata quindi costruita una funzione che lega le due misure, utilizzando non un luxmetro, ma il sensore di luminosità di uno smartphone. È stato prezioso il materiale fornito dal sito AllAboutCircuits[12], comprendente anche un foglio di calcolo che automaticamente crea la funzione di correlazione a partire dai dati forniti.

La funzione risultante è la seguente: $lux = 6207 * resistance^{-0.52}$.

4.2.3 Schema Circuito

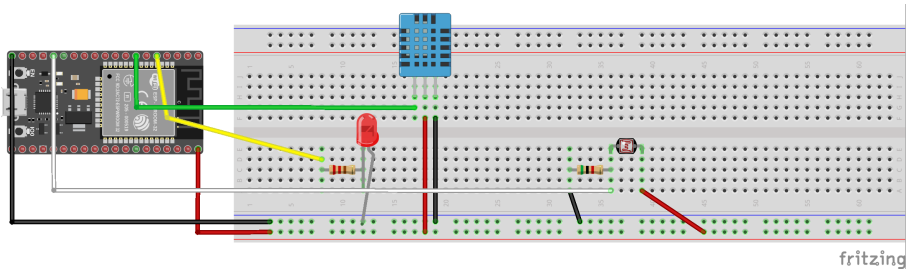

Figura 4.1: Schema elettrico del device

I collegamenti rosso e nero rappresentano rispettivamente la corrente a 5 volt ed il ground.

Il resistore riguardante il sensore di luminosità è da 5KΩ, scelto in base alla luminosità dell'ambiente in cui è collocato il sistema, mentre quello relativo al led è da 220Ω.

Sono stati usati sempre pin digitali, fatta eccezione per il fotoresistore che ne utilizza uno analogico.

4.2.4 Software

Il software di ESP si basa su una architettura piuttosto semplice, essendo i suoi compiti poco complessi.

Ho creato quindi una macchina a stati finiti composta da un solo task.

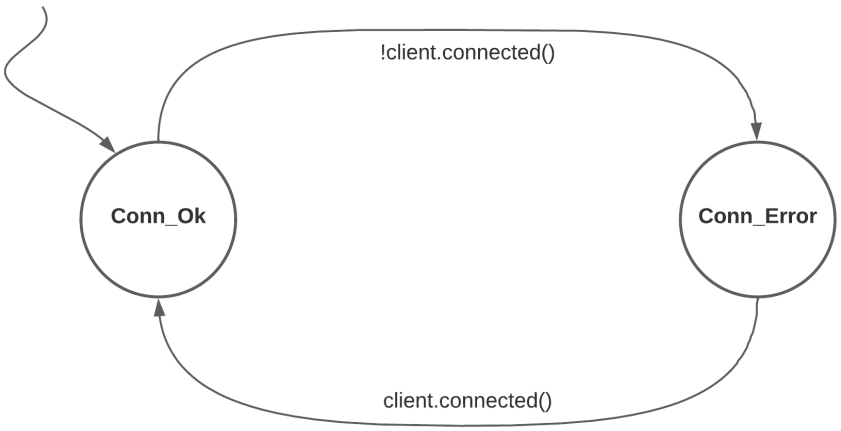

Figura 4.2: Macchina a stati finiti del MainTask

Nel sistema si passa da uno stato all'altro in base alla connessione del client MQTT, come raffigurato nella figura 4.2.

Nello stato *Conn_Ok* il task si occupa di leggere i sensori e inviare i dati attraverso il client MQTT, mentre nello stato *Conn_Error* si occupa di ristabilire la connessione verso il client nel caso in cui esso si disconnetta.

Librerie Utilizzate

All'interno del mio sistema ESP ho utilizzato diverse librerie esterne, quali Arduino Json[13] per la serializzazione e deserializzazione dei JSON, PubSubClient[14] per il client MQTT, ESP32Ping[15] per pingare un host, e la libreria per il DHT11[16].

4.3 Digital Twin

Il sistema si basa sul Digital Twin della serra, creato e gestito tramite la piattaforma Eclipse Ditto e facente uso dello standard Web Of Things.

4.3.1 Modello

Come riportato nella sezione 3.2, la modellazione di un Digital Twin, facendo uso dell'integrazione del WoT su Ditto, parte dalla creazione del Thing Model.

Il Thing Model della serra comprende al suo interno le parti (*tm:submodel*) che poi diventeranno le features all'interno di Ditto, che sono:

- *temperature*: rappresenta la temperatura rilevata in gradi Celsius.
- *humidity*: rappresenta l'umidità rilevata percentuale nell'aria.
- *brightness*: rappresenta la luminosità rilevata in Lux.
- *light*: rappresenta lo stato del LED presente nella serra.

All'interno del Thing Model vengono modellati anche gli attributi del Thing (*SerialNO*) e le azioni e gli eventi che possono essere effettuati/ricevuti.

In particolare, è presente l'azione *switchLight*, che serve per accendere e spegnere il led, e l'evento *high-temperature*, che notifica una temperatura registrata più alta della norma.

È stata usata l'ontologia om-2[10] per rappresentare le unità di misura delle feature. Di seguito riporto il Thing Model della *greenhouse*, e quello della feature *temperature*.

```
{
  "@context": [
    "https://www.w3.org/2022/wot/td/v1.1"
  ],
  "@type": "tm:ThingModel",
  "title": "Green House",
  "description": "A Green House. It has a Humidity/Temperature
     Sensor, a Brightness Sensor, one Light.",
  "version": {
    "model": "1.0.0"
  },
  "links": [
    {
      "rel": "tm:submodel",
      "href": "https://raw.githubusercontent.com/aleshark87/
         WoTModels/main/greenhouse/temperature.tm.jsonld",
      "type": "application/tm+json",
      "instanceName": "temperature"
    },
    {
      "rel": "tm:submodel",
      "href": "https://raw.githubusercontent.com/aleshark87/
         WoTModels/main/greenhouse/humidity.tm.jsonld",
      "type": "application/tm+json",
      "instanceName": "humidity"
    },
    {
      "rel": "tm:submodel",
      "href": "https://raw.githubusercontent.com/aleshark87/
         WoTModels/main/greenhouse/brightness.tm.jsonld",
      "type": "application/tm+json",
      "instanceName": "brightness"
    },
    {
      "rel": "tm:submodel",
      "href": "https://raw.githubusercontent.com/aleshark87/
         WoTModels/main/greenhouse/light.tm.jsonld",
      "type": "application/tm+json",
      "instanceName": "light"
    }
  ],
  "properties": {
    "serialNo": {
```

```
      "title": "Serial number",
      "type": "string"
    }
  },
  "actions": {
    "switchLight": {
      "title": "Switches the Green House Light",
      "description": "Switches the Light of the greenhouse.",
      "type": "boolean"
    }
  },
  "events": {
    "high-temperature": {
      "title": "High Temperature",
      "description": "The actual Temperature is high. Returns the
          temperature",
      "data": {
        "type": "integer"
      }
    }
  }
}
```

Listato 4.1: Thing Model della serra

```
{
  "@context": [
    "https://www.w3.org/2022/wot/td/v1.1",
    {
        "om":
            "http://www.ontology-of-units-of-measure.org/resource/om-2/"
    }
  ],
  "@type": "tm:ThingModel",
  "id": "temperature",
  "title": "Temperature",
  "version": {
    "model": "1.0.0"
  },
  "properties": {
    "value": {
        "description": "Temperature Value",
        "type": "number",
        "minimum": 0,
        "maximum": 50,
        "unit": "om:degree_Celsius"
    }
  }
}
```

Listato 4.2: Thing Model della temperatura

I 4.1 e 4.2, oltre agli altri modelli relativi alle feature, vengono poi "convertiti" da Ditto in Thing Description, fornendo al loro interno gli indirizzi e i protocolli necessari per interagire con il Digital Twin.

4.3.2 Ditto Connection

Allo scopo di stabilire una connessione bidirezionale tra il device e Ditto, è stata usata la parte relativa alle Connections, approfondite nella sezione 2.10.

È stata usata una connessione MQTT, utilizzando come broker quello fornito dall'endpoint pubblico *test.mosquitto.org*[17].

Attraverso le Source vengono ricevuti i messaggi provenienti dal device, mentre attraverso i Target vengono mandati i messaggi verso il device.

Source

Ditto necessita di specificare il topic MQTT per i messaggi in arrivo sulla Source, che in questo caso è *com.greenhouse/#*.

Dal device alla piattaforma possono essere mandati due tipi di messaggi: quello relativo all'aggiornamento dei dati e quello relativo all'emissione di un evento (in questo caso, solamente *high-temperature*).

```
{"thingId":"com.project.thesis:greenhouse01",
"type":"event","temperature":22.5}
```

Listato 4.3: Messaggio di tipo "Event"

```
{"thingId":"com.project.thesis:greenhouse01",
"type":"update","temperature":22.5,
"humidity":76,"brightness":81,"light":"off"}
```

Listato 4.4: Messaggio di tipo "Update"

Allo scopo di mandare messaggi JSON dal device, è stato necessario applicare un mapping al payload dei messaggi, per tradurli da JSON a DittoProtocol. Infatti, nel caso del 4.3 è necessario costruire l'oggetto DittoProtocol per generare messaggi, mentre nel caso di 4.4 vengono generati degli update (modify commands).

Il Mapper è stato implementato in Javascript e una volta che la traduzione avviene, i messaggi che Ditto riceve in DittoProtocol attraverso la Source, modificano il dominio di Ditto.

Target

Il Target serve per mandare messaggi da Ditto verso il device. Anche in questo caso è necessario specificare il topic MQTT, ma questa volta sarà quello a cui si iscriverà il device per ricevere messaggi. Il topic è *"com.greenhouse.notification/thing:id "*. È stato usato soltanto per inoltrare le richieste che giungono dalla applicazione (passando per Ditto) al device, come quella della azione *switchLight*.

Attraverso i *topic*, è possibile specificare quali parti del dominio Ditto si vogliono ricevere, e nel caso della mia applicazione mi sono inscritto solo a quello relativo ai messaggi. Anche in questo caso sarebbe stato possibile applicare un mapping al payload, ma non è stato ritenuto necessario visto che il DittoProtocol è un messaggio in JSON e decodificarlo non è difficile come codificarlo.

4.4 Applicazione Web

Lo scopo della applicazione web è quello di visualizzare i dati del Digital Twin in tempo reale, e di offrire la possibilità di effettuare azioni (*switchLed*), e di ricevere gli eventi (*high-temperature*).

4.4.1 Tecnologie Utilizzate

L'applicazione è stata creata utilizzando HTML e Javascript.

In particolare, librerie come Bootstrap, JQuery, JsonPath-Plus ed Ajax sono state utilizzate. Bootstrap è stato utilizzato per costruire l'interfaccia, di cui parlerò nella prossima sezione, mentre JQuery per interagire con gli oggetti del dominio in maniera più semplice e snella.

JsonPath-Plus è invece utile per accedere agli oggetti JSON, e in particolare è stato usato per le Thing Description. Il ruolo di Ajax è quello di effettuare richieste HTTP, utili per interagire col dominio Ditto.

4.4.2 Interfaccia

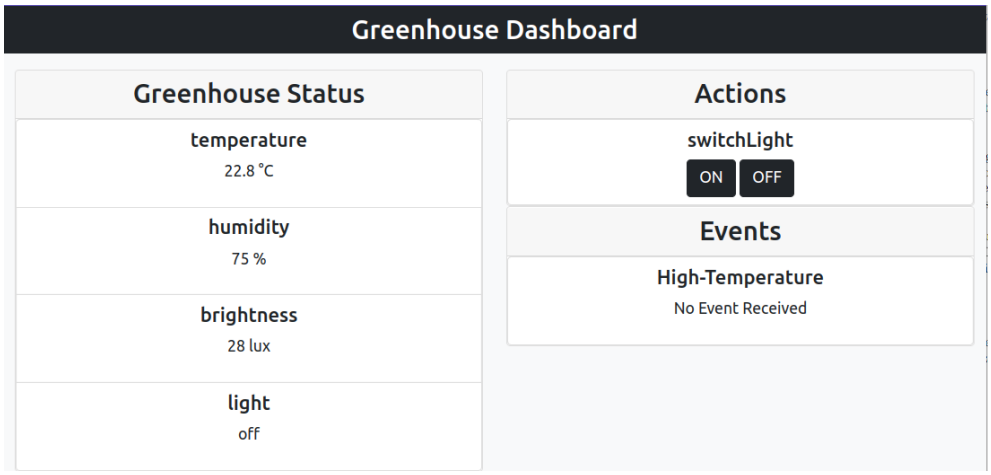

Figura 4.3: Interfaccia dell'applicazione

La figura 4.3 è l'interfaccia utente dell'applicazione.

Sulla sinistra, sono presenti i dati del Digital Twin, raggruppati per feature, mentre sulla destra, le azioni e gli eventi presenti.

L'interfaccia viene costruita dinamicamente, in base a quante proprietà, azioni ed eventi vengano trovati.

4.4.3 Utilizzo della Thing Description

Aspetto cruciale dell'implementazione della Web App è quello relativo all'utilizzo delle Thing Description.

Come specificato nel capitolo dedicato al Web Of Things (3), le Thing Description sono pensate per essere usate dalle applicazione che fanno uso dei Digital Twin.

La prima scelta è ricaduta su progetti facenti uso della WoT Scripting API (3.1.4) come Eclipse Thingweb[8].

Il progetto permette di "consumare" una Thing Description, esponendone le sue proprietà, azioni ed eventi attraverso la sua API. È inoltre la referenza ufficiale per quanto riguarda il Web of Things.

Problemi nell'utilizzo di Eclipse Thingweb con Ditto

Purtroppo però, dato che l'integrazione di Ditto con il WoT è agli albori, e anche il progetto Eclipse Thingweb è relativamente giovane, non sono riuscito ad utilizzarlo per utilizzare Thing Description provenienti da Ditto.

Ho comunque avvertito il team di sviluppo con un Github Issue[18], e anche loro mi hanno confermato che ci sono delle parti, nella Thing Description di Ditto, che non sono state ancora implementate.

Credo che sarebbe possibile creare un work-around, ma purtroppo, non aiutato dalla mia scarsa esperienza con Typescript, ho preferito passare oltre.

Utilizzo della Thing Description con la mia implementazione

Ho comunque cercato, per quanto possibile, di utilizzare la sezione *forms* della Thing Description, allo scopo di ottenere in maniera dinamica gli endpoint necessari per ottenere le proprietà, le azioni e gli eventi.

Questo è stato fatto tramite la libreria JsonPath-Plus, che mi ha consentito di navigare attraverso il JSON in maniera veloce e semplice.

Ho avuto problemi soltanto con l'endpoint relativo agli eventi: Ditto fornisce soltanto l'endpoint per registrarsi tramite Server Sent Events[5], ma, nonostante sia riuscito senza problemi a registrarmi a quelli relativi alle proprietà del Thing, non sono riuscito ad utilizzare quelli relativi ai Messaggi Ditto (con cui sono implementati gli Eventi). Anche in questo caso ho riportato al team di sviluppo il problema.

Per ovviare al problema, ho utilizzato la connessione a Ditto tramite WebSocket, registrandomi soltanto ai messaggi.

4.4.4 Protocolli utilizzati

Per quanto riguarda le proprietà del Thing (features), Ditto (nella Thing Description) offre la possibilità di recuperarle sia tramite HTTP API, che tramite i SSE[5].

Ho scelto di utilizzare un approccio ibrido, utilizzando le HTTP API per recuperare le informazioni del Digital Twin appena il sito si avvia, e i SSE per gestire gli update del modello.

I Server Sent Events infatti garantiscono all'applicazione di essere responsiva agli update, togliendo allo sviluppatore la necessità di gestire aggiornamenti periodici: inoltre, si lega in maniera migliore con la natura asincrona di Javascript.

L'azione consiste in un messaggio inviato nella *inbox* di un Thing, e viene utilizzata una chiamata all'HTTP API.

Invece, come detto in precedenza, gli eventi utilizzano la WebSocket per essere notificati.

Conclusioni

Il corso di Sistemi Embedded è stato uno dei miei preferiti, e lo studio da me effettuato sui Digital Twin lo ha sicuramente arricchito molto, e reso molto più di valore.

Sono sicuro che negli anni a venire potranno essere ancora più sviluppati, e ancora più utili per quello che per me deve essere lo scopo dell'informatica: rendere la nostra qualità di vita migliore.

Il framework da me esplorato, Eclipse Ditto, si è rivelato un buon progetto, anche se in alcuni parti possono essere apportate delle migliorie. Inoltre, ho potuto confrontarmi spesso col team di sviluppo quasi in tempo reale, e questo per me è stato fondamentale per capirne al meglio le sue parti.

A mio parere, uno sviluppo interessante potrebbe essere l'integrazione con un qualche database come IoTDb [19], allo scopo di mantenere anche lo storico dei dati di un Digital Twin.

Infatti in questo modo, sarebbe anche possibile fare studi utilizzando il machine learning, allo scopo di prevedere comportamenti dei Digital Twin (una delle parti fondamentali elencate nel Capitolo 1).

Anche il Web of Things è secondo me un buon progetto, e soprattutto, è in piena fase di sviluppo, potendo essere potenzialmente per i prossimi anni uno standard molto importante non solo per i Digital Twin, ma per tutto il mondo dell'Internet Of Things.

Il fatto che ci sia dietro il W3C è un ottimo presupposto per il suo futuro, essendo già leader per quando riguarda gli standard Web. Inoltre, il loro processo di sviluppo è completamente aperto, rendendolo aperto a proposte e revisioni da parte di tutta la community.

Sviluppare un vero sistema basato su un Digital Twin è stato fondamentale per capire meglio le sue parti, ma anche la sua filosofia.

Sono generalmente soddisfatto del lavoro svolto, ma anche sicuro di poter fare di meglio: ad esempio, mi sarebbe piaciuto essere riuscito ad utilizzare la libreria Eclipse Thingweb[8] per usare una Thing Description ma sono sicuro che nei mesi/anni a venire i due progetti (Ditto e Thingweb) potranno lavorare meglio sulla loro integrazione, facendo di Eclipse IoT[20] un ecosistema sempre più completo.

Ringraziamenti

La tesi è stata la conclusione di un percorso che per me è stato incredibile. Vorrei ringraziare fin da subito il Prof. Alessandro Ricci, che mi ha seguito fin dal tirocinio curricolare e mi ha fatto scoprire l'argomento dei Digital Twin, dandomi la possibilità di imparare dai miei errori e di capirlo al meglio.

Una menzione va a tutti i miei amici, quelli che se ne sono andati, e quelli che sono rimasti. Parlo degli amici che conosco da tanti anni, quelli con cui esco da prima che scegliessi questa università, che hanno rappresentato per me una sicurezza nei momenti in cui ero stanco dallo studio, e magari arrabbiato per qualche bug.

I miei amici d'università, nonostante la pandemia globale che tutti conosciamo ci abbia fatto perdere tanto tempo insieme, sono stati se vogliamo ancora più importanti. Posso dire con sicurezza che non sarei mai arrivato qui senza il loro aiuto nella preparazione di molti esami, ma soprattutto senza l'ispirazione che mi hanno sempre dato, a lavorare duro e a non mollare.

La mia famiglia è però alla base di tutto il mio percorso, per tutti i sacrifici che hanno fatto per permettermi di studiare, e per la fiducia che nei miei confronti, nonostante a volte sarebbe dovuta mancare, non ha mai vacillato.

Vorrei fare una menzione anche per mio fratello Simone, che per me ha rappresentato l'inizio della mia passione per l'informatica. Ricordo con gioia quando io bambino, e lui adolescente passavamo ore insieme davanti al computer, e mi insegnava ad utilizzarlo. Lui, ormai ingegnere meccanico, mi ha sempre spronato a fare meglio, e almeno nell'informatica, conto di averlo superato.

Bibliografia

[1] Roberto Minerva, Gyu Myoung Lee e Noel Crespi. "Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models". In: *Proceedings of the IEEE* PP (giu. 2020), pp. 1–40. DOI: 10.1109/JPROC.2020.2998530 (cit. a p. 1).

[2] Michael Grieves. "Product Lifecycle Management: Driving the Next Generation of Lean Thinking". In: (gen. 2005) (cit. a p. 1).

[3] Sebastian Haag e Reiner Anderl. "Digital Twin – Proof of Concept". In: *Manufacturing Letters* 15 (feb. 2018). DOI: 10.1016/j.mfglet.2018.02.006 (cit. a p. 1).

[4] Qin Wang et al. *Non-Fungible Token (NFT): Overview, Evaluation, Opportunities and Challenges*. Mag. 2021 (cit. a p. 4).

[5] *Server Sent Events*. URL: https://html.spec.whatwg.org/multipage/server-sent-events.html (cit. alle pp. 23, 44, 45).

[6] W3C. *Web Of Things - Digital Twin Use Case*. 2020. URL: https://www.w3.org/TR/wot-architecture/#digital-twins= (cit. a p. 29).

[7] Manu Sporny, Gregg Kellogg e Markus Lanthaler. "JSON-LD 1.0 - A JSON-based Serialization for Linked Data". In: *W3C Recommendation* (gen. 2014) (cit. a p. 30).

[8] *Eclipse Thingweb*. 2020. URL: https://projects.eclipse.org/projects/iot.thingweb (cit. alle pp. 32, 44, 47).

[9] Laura Daniele et al. "A SAREF Extension for Semantic Interoperability in the Industry and Manufacturing Domain". In: ott. 2018, pp. 201–207. ISBN: 9781786303738. DOI: 10.1002/9781119564034.ch25 (cit. a p. 33).

[10] Hajo Rijgersberg, Mark Assem e Jan Top. "Ontology of units of measure and related concepts". In: *Semantic Web* 4 (gen. 2013), pp. 3–13. DOI: 10.3233/SW-2012-0069 (cit. alle pp. 33, 38).

[11] *ESP32*. URL: https://www.espressif.com/en/products/socs/esp32 (cit. a p. 35).

[12] *Design a Luxmeter using a Light Dependent Resistor*. URL: https://www.allaboutcircuits.com/projects/design-a-luxmeter-using-a-light-dependent-resistor/ (cit. a p. 36).

[13] *Arduino Json*. URL: https://arduinojson.org/ (cit. a p. 38).

[14] *PubSubClient*. URL: https://pubsubclient.knolleary.net/ (cit. a p. 38).

[15] *ESP32Ping*. URL: https://github.com/marian-craciunescu/ESP32Ping (cit. a p. 38).

[16] *DHT-sensor-library*. URL: https://github.com/adafruit/DHT-sensor-library (cit. a p. 38).

[17] *Mosquitto public broker*. URL: http://test.mosquitto.org (cit. a p. 41).

[18] *Ditto Descriptions with ThingWeb Issue*. URL: https://github.com/eclipse/thingweb.node-wot/issues/745 (cit. a p. 44).

[19] *IoTDb*. URL: https://iotdb.apache.org/ (cit. a p. 47).

[20] *Eclipse IoT*. URL: https://iot.eclipse.org/ (cit. a p. 47).