ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

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

SVILUPPO E GESTIONE DI DIGITAL TWIN: STUDIO E CONFRONTO DEI FRAMEWORK ECLIPSE DITTO E WLDT

Elaborato in
SISTEMI EMBEDDED E INTERNET-OF-THINGS

Relatore
Prof. ALESSANDRO RICCI

Corelatore
Prof. MARCO PICONE
Dott. SAMUELE BURATTINI

Presentata da
FABIO VEROLI

Anno Accademico 2022 – 2023

"Every piece of information in the world has been copied, backed up, except the human mind. The last analog device in a digital world." - Dr. Robert Ford (Anthony Hopkins), Westworld

Indice

Introduzione **vii**

1 Panoramica sui Digital Twin **1**
1.1 Origini del Termine e Prime Definizioni 1
1.2 Successive Definizioni e Sviluppi 3
1.2.1 Proprietà Caratterizzanti dei DT nel contesto IoT 4
1.3 Principali Tecnologie e Framework per Digital Twin 9
1.4 Analisi e Confronto dei Framework per Digital Twin 10

2 Analisi del framework Eclipse Ditto **11**
2.1 Architettura 12
2.2 Entità del Modello 14
2.2.1 Thing 14
2.2.2 Feature 15
2.2.3 Policy 15
2.3 Signal 16
2.3.1 Change Notification 18
2.4 Ditto Protocol 18
2.4.1 Specifiche del protocollo 19
2.4.2 Protocol Topic 20
2.5 Message 20
2.6 Connection 22
2.6.1 Source 23
2.6.2 Target 24
2.6.3 Payload Mapping 24
2.7 Compatibilità con W3C Web of Things 24
2.7.1 W3C Web of Things 25
2.7.2 Integrazione con Eclipse Ditto 25
2.8 API 27

3 Analisi del framework White Label Digital Twin **29**
3.1 Astrazione e Modello Digital Twin 30

3.2 Processo di Shadowing 32
3.3 Ciclo di vita del Digital Twin 32
3.3.1 Da Unbound a Bound 33
3.3.2 Da Bound a Shadowed 34
3.4 Struttura della Libreria 35

4 Sviluppo dei Digital Twin utilizzando i due framework 37
4.1 Caso di studio: Smart Bridge 37
4.1.1 Specifiche 37
4.1.2 Modello Digital Twin 39
4.1.3 Implementazione 42
4.1.4 Dashboard Web 45
4.2 Sviluppo di Digital Twin utilizzando Eclipse Ditto 46
4.2.1 Policy 47
4.2.2 Thing 48
4.2.3 Connection 51
4.3 Sviluppo di Digital Twin utilizzando WLDT 54
4.3.1 Physical Adapter 54
4.3.2 Shadowing Function 55
4.3.3 Digital Adapter 57
4.3.4 Processo del Digital Twin 59

5 Confronto Finale tra i due framework 61
5.1 Architetture a confronto 61
5.2 Modellazione del Digital Twin a confronto 62
5.3 Interazione con il Physical Asset a confronto 63
5.4 Sviluppo del Digital Twin a confronto 64

Conclusioni 67

Ringraziamenti 69

Introduzione

I **Digital Twin** sono un paradigma che ha riscosso notevole interesse negli ultimi anni, sia nel contesto accademico che in quello industriale. Questo modello si basa sull'idea che si possa creare un costrutto informativo digitale di un sistema fisico come un'entità autonoma. L'informazione digitale è un *"gemello"* della controparte fisica e rimarrà connessa con il sistema fisico per tutto il suo ciclo di vita. In altre parole, un Digital Twin fornisce la capacità di replicare una risorsa fisica in una controparte software, che riflette le stesse proprietà e caratteristiche dell'oggetto originale all'interno di uno specifico contesto applicativo.

Uno degli obiettivi fondamentali dei Digital Twin è facilitare l'interoperabilità dei sistemi fisici, creando uno strato di astrazione digitale che permetta loro di comunicare con altri sistemi fisici o con applicazioni esterne. Inoltre, i Digital Twin possono ampliare le funzionalità del sistema fisico, ottimizzandone le proprietà e i comportamenti.

Numerose grandi aziende si stanno interessando alla tecnologia dei Digital Twin, sviluppando soluzioni proprietarie che, tuttavia, generano una mancanza di uniformità nella modellazione dei Digital Twin e riducono l'interoperabilità tra i sistemi.

L'obiettivo di questa tesi è confrontare due framework open-source (Eclipse Ditto e WLDT) al fine di evidenziare come gestiscono la modellazione dei Digital Twin e l'interazione con i sistemi fisici. Questo confronto, oltre a individuare i punti di forza e le limitazioni di ciascun framework, potrebbe fornire spunti utili per definire un approccio generale alla modellazione dei Digital Twin e alla gestione della comunicazione con i sistemi fisici.

La tesi è strutturata in cinque capitoli: i primi tre capitoli offrono una panoramica generale sui Digital Twin e sui due framework in esame, mentre i successivi due seguono le fasi di sviluppo del Digital Twin prima di passare al confronto finale tra i due framework. Nello specifico, il primo capitolo introduce l'evoluzione del concetto di Digital Twin, identificando una serie di proprietà derivanti da diversi ambiti di applicazione. Il secondo e terzo capitolo introducono rispettivamente **Eclipse Ditto** e **WLDT**, i due framework open-source oggetto di confronto. Il quarto capitolo esamina le scelte fatte durante

la creazione dei Digital Twin, partendo dall'implementazione del sistema fisico e procedendo alla selezione delle proprietà rilevanti e delle funzionalità da utilizzare per modellare il Digital Twin, per concludere con lo sviluppo dei Digital Twin utilizzando i due framework e argomentando le scelte fatte. Infine, il quinto capitolo si focalizza sul confronto tra le due tecnologie, analizzando le differenze architetturali, la modellazione e lo sviluppo dei Digital Twin.

Capitolo 1

Panoramica sui Digital Twin

1.1 Origini del Termine e Prime Definizioni

Il concetto di **Digital Twin** (**DT**) appare per la prima volta in una presentazione destinata all'industria presso l'Università del Michigan nel 2002, in occasione della creazione di un centro di **Product Lifecycle Management** (**PLM**). La diapositiva della presentazione (Figura 1.1), ideata dal Dr. Grieves, non introduce direttamente il concetto come lo conosciamo oggi, ma propone comunque un "*Conceptual Ideal for PLM*", in cui erano già contenuti tutti gli elementi di un Digital Twin: lo spazio reale, lo spazio virtuale, il collegamento per il flusso di dati dallo spazio reale allo spazio virtuale, il collegamento per il flusso di informazioni dallo spazio virtuale a quello reale e ad eventuali sottospazi virtuali.

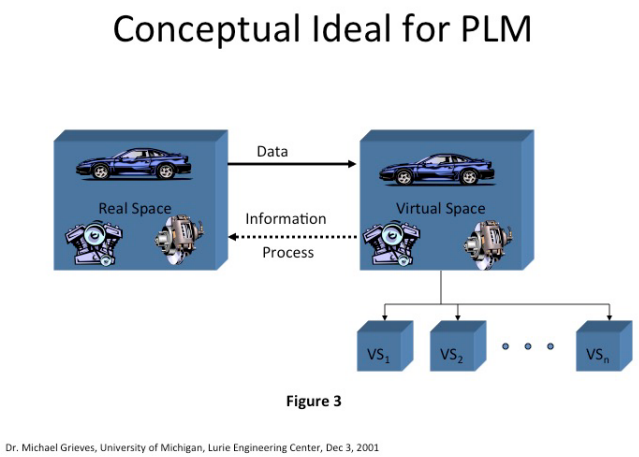


Figura 1.1: Diapositiva del Dr. Grieves (2002)

Il concetto di PLM viene ulteriormente sviluppato in [4], dove per la prima volta si utilizza il termine Digital Twin per riferirsi al modello impiegato. Il PLM, come si evince dal termine, non è una rappresentazione statica, ma piuttosto un collegamento tra il sistema fisico e quello virtuale durante tutte le fasi del ciclo di vita del prodotto (creazione, produzione, esercizio e dismissione).

Il Dr. Grieves fornisce quindi una prima definizione di **Digital Twin** [5]: un DT è un insieme di costrutti informativi virtuali che descrive completamente un prodotto fisico potenziale o effettivo, dal livello micro atomico al livello geometrico macro. Nelle sue condizioni ottimali, qualsiasi informazione ottenibile ispezionando un prodotto manufatto fisico può essere ottenuta dal suo DT.

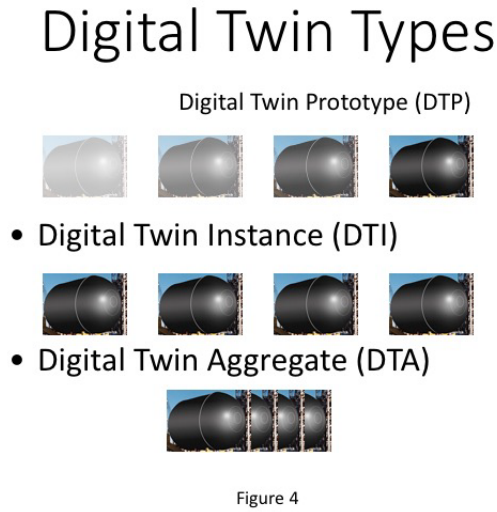


Figura 1.2: Tipi di Digital Twin

Come illustrato in figura 1.2, vengono definiti vari tipi di DT:

- **Digital Twin Prototype** (**DTP**): è una descrizione del prototipo del prodotto fisico, contiene tutte le informazioni necessarie a descrivere e produrre la versione fisica che duplicherà quella virtuale.
- **Digital Twin Instance** (**DTI**): descrive uno specifico prodotto fisico a cui un singolo DT rimarrà collegato per l'intero ciclo di vita.
- **Digital Twin Aggregate** (**DTA**): è un'aggregazione di tutte le DTI. A differenza delle DTI, potrebbe non avere una struttura dati indipendente, ma potrebbe essere un costrutto computazionale che ha accesso a tutte le DTI e le interroga ah-hoc o pro-attivamente.

Nella suo primo utilizzo nel campo della produzione industriale, il Digital Twin è concepito come un modello dinamico che accompagna il prodotto durante tutto il suo ciclo di vita. Emergono due proprietà fondamentali di un DT, che deve essere predittivo ed interrogativo, cioè rispettivamente: essere in grado di prevedere comportamenti e prestazioni future ed essere interrogabile sulla storia corrente o passata del sistema. Il modello del DT è dinamico in quanto svolge un compito diverso in ogni fase del ciclo di vita del prodotto:

- Nella fase di creazione, il sistema virtuale (DTP) è l'unico esistente, non vi è ancora una controparte fisica. Il prototipo virtuale, utilizzato in diversi spazi virtuali, consente di sottoporre il sistema a test distruttivi a costo zero. Il DTP consente non solo di verificare e validare i requisiti, detti predicted desirable, eliminare problemi e fallimenti, detti predicted undesirable, ma fornisce anche l'opportunità di identificare gli unpredicted undesirable testando diverse configurazioni con diversi utenti. Una volta che il sistema è completo e validato, le informazioni ricavate dal DTP potranno poi essere utilizzate per creare il prodotto fisico nel mondo reale.

- Nella fase di produzione il prodotto fisico viene costruito concretamente, verrà quindi creata una DTI che permetta di modellare tutte le configurazioni, i dati e le informazioni del prodotto in uno spazio virtuale.

- Nella fase di assistenza/manutenzione si può verificare se le predizioni effettuate sul sistema si riveleranno corrette. Il prodotto fisico e quello virtuale mantengono il collegamento bidirezionale, che consente all'oggetto fisico di notificare i propri cambiamenti e a quello virtuale di prevedere prestazioni e fallimenti.

- Nella fase di dismissione, il prodotto fisico viene ritirato, ma potrebbe essere utile mantenerne la controparte virtuale per utilizzi futuri dei dati, come ad esempio trarne informazioni utili per i prodotti di generazioni future.

1.2 Successive Definizioni e Sviluppi

Dalla sua prima formulazione, il concetto di Digital Twin ha suscitato un crescente interesse sia in ambito industriale che in quello accademico. Dopo la sua iniziale applicazione nel settore della produzione industriale, l'uso dei DT si è esteso anche all'Internet of Things (IoT) e ai Cyber Physical System (CPS). Nell'ambito dell'IoT, un Digital Twin può assumere un ruolo versatile,

consentendo non solo la creazione di applicazioni IoT, ma possedendo anche la capacita di rilevare ed attuare, tipica delle tecnologie IoT.

L'introduzione del concetto in ambiti diversi da quello industriale ha portato a una ridefinizione, allargando il suo significato da un prospettiva strettamente industriale e legata al prodotto a una che può modellare qualsiasi tipo di oggetto fisico, compresi quelli intangibili. Da questo punto in avanti, ci riferiremo alla controparte fisica del Digital Twin come "oggetto fisico" o "risorsa fisica" o "Physical Asset" (PA).

A fronte di queste considerazioni, forniamo la definizione aggiornata [7]:

Un **Digital Twin** (DT) è una rappresentazione software esaustiva di un singolo Physical Asset (PA). Include le proprietà, le condizioni, le relazioni e i comportamenti della risorsa del mondo reale tramite modelli e dati.

Un **Digital Twin** è un insieme di modelli realistici che possono digitalizzare il comportamento di una risorsa nell'ambiente in cui è implementata.

Il **Digital Twin** rappresenta e riflette il proprio Physical Twin e rimane la sua controparte virtuale per l'intero ciclo di vita della risorsa.

Queste definizioni ampliate riflettono l'evoluzione del concetto e la sua applicazione in contesti sempre più diversificati, fornendo una base concettuale robusta per il proseguimento dell'analisi e dello studio dei Digital Twin.

1.2.1 Proprietà Caratterizzanti dei DT nel contesto IoT

Dopo aver fornito la definizione di Digital Twin, è essenziale individuare e definire una serie di proprietà fondamentali dei DT che coinvolgano diversi contesti e situazioni, ma rimangano allo stesso tempo il più generali possibili [7].

Alcune proprietà sono fondamentali per il funzionamento del Digital Twin, senza esse non sarebbe possibile una sua implementazione, mentre altre estendono e arricchiscono il valore del Digital Twin. Sarà altresì importante garantire la validità di queste proprietà.

Representativeness e Contextualization

Il Digital Twin dovrebbe essere il più possibile fedele al PA originale. Deve essere supportato da un modello progettato ed implementato con specifici obbiettivi e contesti operativi. Uno dei concetti alla base dei DT è quanto la replica sia rappresentativa dell'oggetto originale. La **representativeness** viene valutata sotto tre aspetti principali:

- **Similarità**: quanto il Digital Twin riproduce fedelmente l'oggetto originale nei suoi stati e nelle sue proprietà.
- **Casualità**: è la probabilità che il Digital Twin presenti uno stato o delle proprietà divergenti rispetto al PA.
- **Contestualizzazione**: le proprietà precedenti devono essere considerate nel contesto operativo del PA. Se il Digital Twin opera in un certo ambiente, solo un sottoinsieme di proprietà, funzioni ed informazioni del PA sarà rilevante.

La **contextualization** richiede che tutte le caratteristiche rilevanti e i dati disponibili siano necessari e sufficienti per rappresentare l'oggetto fisico nello spazio virtuale specifico preso in considerazione.

Reflection

Il concetto di **reflection** si riferisce alla rappresentazione accurata e tempestiva da parte del Digital Twin di tutti gli attributi, le caratteristiche, gli eventi, le azioni e gli stati che caratterizzano una risorsa fisica e viceversa.

La reflection sottolinea la necessità per il modello digitale di essere rappresentativo della risorsa fisica, riflettendo fedelmente attributi e stato dell'oggetto fisico. In questo contesto, la risorsa deve essere situata nello spazio e nel tempo, e tutti gli attributi devono essere una tripletta di timestamp, posizione e valore.

Replication

Indica la capacità generale di replicare un oggetto in un ambiente diverso. Una risorsa fisica può essere virtualizzata e replicata molteplici volte all'interno di uno spazio virtuale; a sua volta, anche la risorsa digitale può essere clonata più volte. Fondamentale è che nel corso del tempo l'insieme di attributi che caratterizzano la risorsa fisica e la risorsa digitale replicata rimangano consistenti.

La replicabilità viene garantita dall'abilità degli ambienti software di virtualizzare i componenti. La virtualizzazione può, ad esempio, alleggerire il carico di richieste inviate ad un PA, reindirizzandole a uno o più Digital Twin, che replicano la risorsa, in esecuzione sul cloud. I vari Digital Twin possono poi cooperare tra di loro scambiandosi informazioni e dati sul PA.

Entanglement

All'interno della nozione di Digital Twin è implicito il collegamento tra risorsa fisica e digitale. Ciò comporta che tutte le informazioni che descrivono

un oggetto debbano essere trasferite, (quasi) in tempo reale, alla replica digitale in modo da renderle disponibili ad applicazioni e servizi.

Questa comunicazione è definita **entanglement**, poiché si riferisce allo scambio istantaneo di informazioni tra due entità strettamente collegate. Consideriamo tre proprietà per definire questa comunicazione:

- **Connettività**: deve esistere un modo diretto o indiretto per comunicare tra il PA e il DT, eventuali cambiamenti di stato o nei dati. La trasmissione è diretta se la risorsa e la replica possono comunicare direttamente tra di loro, mentre è indiretta se si affidano a terzi per inviare o ricevere informazioni.

- **Prontezza**: lo scambio di informazioni tra PA e DT deve essere tempestivo, in modo che il tempo trascorso tra le variazioni di stato del PA sia trascurabile rispetto all'utilizzo previsto del DT da parte di applicazioni e utenti. In linea generale, possiamo considerare il tempo che intercorre tra due variazioni di stato come il limite superiore per inviare aggiornamenti al DT.

- **Associazione**: la relazione tra PA e DT può essere unidirezionale o bidirezionale. Solitamente si tratta di una relazione unidirezionale in cui le informazioni vanno dalla risorsa fisica a quella digitale, ma una comunicazione bidirezionale consentirebbe al Digital Twin di fornire aggiornamenti utili per migliorare le funzionalità del PA.

Si parla di entanglement forte se la connessione è costante, in tempo reale e bidirezionale; in questo caso, anche il DT può modificare lo stato del PA. Mentre si indica con entanglement debole una comunicazione non in tempo reale che potrebbe essere interrotta per un certo periodo di tempo.

Persistency

Un Digital Twin deve essere persistente nel corso del tempo. Non tutti i PA sono in grado di fornire in modo continuativo i propri servizi; il Digital Twin deve quindi cercare di mitigare o compensare questa limitazione per consentire un accesso continuo alle informazioni della risorsa.

Il Digital Twin deve essere resiliente e persistente in modo che anche in caso di malfunzionamento del PA, sia in grado di ristabilirne le funzionalità e sincronizzarlo ad uno stato accettabile.

Memorization

Un'altra importante proprietà dei Digital Twin è l'abilità di memorizzare e rappresentare tutti i dati (presenti o passati) rilevanti del PA. Si pone il

problema di selezionare le proprietà e le informazioni rilevanti da memorizzare. Nel contesto dei Digital Twin, si ricorre solitamente ai concetti di abbondanza e completezza, cioè si cerca di memorizzare più dati possibili per catturare tutte le proprietà e le relazioni del PA nell'ambiente in cui opera.

In questo modo, il Digital Twin non solo è in grado di operare nello stato corrente fornendo dati alle applicazioni che li richiedono, ma anche di prevedere future evoluzioni del PA utilizzando quelle proprietà che nel momento della memorizzazione possono sembrare irrilevanti, ma che in future evoluzioni diventeranno essenziali.

Composability

La **composability** per un Digital Twin è la capacità di aggregare diversi oggetti in un sistema composto, permettendo di osservare il comportamento di quest'ultimo così come delle sue singole componenti.

Questo concetto può essere applicato a diversi livelli di granularità a seconda del contesto di applicazione. Ad esempio, ogni sotto-entità di un sistema composto potrebbe essere considerata come un Physical Asset e quindi mappata in un Digital Twin. Tuttavia, il DT che rappresenta l'entità composta dovrebbe comunque essere rappresentato, considerato e vi si dovrebbe poter interagire come un'entità singola.

Per sistemi di dimensioni maggiori, potrebbe essere necessario avere solo una panoramica generale dell'oggetto composto, senza necessariamente accedere alle sotto-entità aggregate. In questo caso, la strutturazione del DT potrebbe essere adattata in modo diverso.

Accountability/Manageability

La proprietà di **accountability/manageability** si riferisce alla capacità di gestire accuratamente i Digital Twin. Mentre i PA possono avere guasti, i corrispondenti DT devono entrare in uno stato di recupero, in modo da essere in grado di rispondere ad interrogazioni sul PA e mostrarne gli ultimi valori funzionali.

I Digital Twin devono fungere da affidabili registri di tutti gli stati di un PA, permettendo il recupero di informazioni in caso di guasto. La gestione e le responsabilità del PA devono essere completamente replicate nel DT, garantendo la sua esistenza oltre la durata di vita della risorsa fisica.

Augmentation

Il PA ha delle funzionalità fisse per l'intero ciclo di vita. L'**augementation** si riferisce alla capacità del DT di migliorare il PA attraverso la demateria-

lizzazione del software, ovvero l'abilità del DT di aggiornare, modificare e potenziare le proprie funzionalità nel tempo.

Questo potenziamento avviene attraverso l'implementazione di nuove funzioni software basate su API o sull'analisi di serie di dati correlati. Le API consentono l'interoperabilità tra PA e DT.

L'augmentation può essere realizzata utilizzando i dati del PA ed esponendo API per controllare, governare, orchestrare o interrogare il DT.

Ownership

L'**ownership** è una proprietà spesso trascurata ma significativa del DT e si manifesta in due modi. In primo luogo, riguarda il possesso dei dati prodotti dal DT, che necessita di regolamentazione. In secondo luogo, si riferisce al possesso del DT stesso, che non necessariamente ha lo stesso possessore del PA. Il settore industriale è interessato ai cambiamenti di possesso e alla loro gestione per motivi commerciali. La questione del possesso può essere complessa, ma la proliferazione dei DT richiede di affrontare questa proprietà in modo da introdurre flessibilità nella rappresentazione logica del PA originale.

Servitization

La **servitization** si riferisce alla capacità di offrire sul mercato un prodotto associandolo a servizi, funzionalità e accesso ai dati mediante strumenti software. I DT, fornendo nuove funzionalità per il PA, generano un alto grado di servitization. La servitization è di grande interesse per il mondo industriale, in quanto la trasformazione da risorsa ad insieme di servizi cambia la percezione del prodotto da parte del cliente. Si potrebbe quindi passare ad un'economia basata su un modello di pagamento per i servizi offerti da una risorsa, anziché per il suo possesso.

Predictability

Il DT rappresenta grandi quantità di dati relativi a eventi e proprietà, operando in contesti noti e ben compresi. La proprietà di **prevedibilità** si riferisce alla possibilità di incorporare un DT in un ambiente specifico e simulare il suo comportamento e le interazioni con altri oggetti in futuro o durante un periodo di tempo specifico. Questo indica la necessità di sviluppare sistemi complessi dedicati alla previsione del comportamento dei DT in contesti specifici.

1.3 Principali Tecnologie e Framework per Digital Twin

Con la crescente adozione del concetto di Digital Twin, numerose aziende hanno sviluppato soluzioni mirate alla modellazione e allo sviluppo dei Digital Twin, al fine di posizionarsi come leader nel settore. Tuttavia, la diversità di approcci adottati da gruppi di ricerca e aziende ha portato a una frammentazione delle soluzioni, spesso caratterizzate da standard proprietari che minano i principi di interoperabilità e flessibilità auspicati nell'implementazione dei Digital Twin. Di seguito, sono presentati alcune delle principali soluzioni proposte da diverse aziende e istituzioni:

- **Microsoft Azure Digital Twins**: piattaforma cloud che permette la creazione e la gestione di Digital Twin, integrando modellazione avanzata e servizi di intelligenza artificiale.
- **Siemens Digital Twin**: orientato all'automazione e alla digitalizzazione industriale, offre strumenti per modellare impianti industriali, macchinari e processi, con un focus particolare sulla produzione.
- **IBM Watson IoT Platform**: fornisce servizi per lo sviluppo di soluzioni IoT, inclusi Digital Twin. Supporta l'analisi dei dati in tempo reale e la gestione dei dispositivi IoT.
- **General Electric**: offre strumenti per la creazione di Digital Twin utilizzati soprattutto nel settore manifatturiero e dell'energia.
- **AWS IoT TwinMaker**: IoT TwinMaker di Amazon Web Services fornisce strumenti per modellare e implementare Digital Twin, integrando diverse risorse IoT.
- **Eclipse Ditto**: progetto open-source che offre una piattaforma per la creazione e la gestione di Digital Twin, consentendo la modellazione flessibile di dispositivi e risorse fisiche.
- **White Label Digital Twin**: progetto accademico open-source che propone un framework per la creazione di Digital Twin come sistemi indipendenti, eseguibili sia nel cloud che in locale.

La scelta del framework più adatto dipenderà dalle specifiche esigenze del settore in cui verrà implementato il Digital Twin. Nei seguenti capitoli 2 e 3, saranno approfonditi i due framework open-source Eclipse Ditto e White Label Digital Twin, analizzandone il funzionamento e le modalità di modellazione dei Digital Twin.

1.4 Analisi e Confronto dei Framework per Digital Twin

Dopo aver fornito una panoramica generale sui Digital Twin, evidenziando le potenzialità e le proprietà offerte da questo paradigma, nei capitoli successivi approfondiremo gli obbiettivi della presente tesi.

Lo scopo di questo progetto di tesi è analizzare e confrontare due framework open-source, Eclipse Ditto e White Label Digital Twin, al fine di evidenziare i punti di forza e le limitazioni di ciascun di essi, nonché i diversi approcci utilizzati per lo sviluppo e la gestione dei Digital Twin. Tale confronto potrebbe poi fornirne spunti interessanti per una futura evoluzione del framework accademico White Label Digital Twin.

Inizieremo analizzando nel dettaglio ciascuno dei due framework (Capitoli 2 e 3), al fine di comprendere come essi modellino i Digital Twin, gestiscano l'interazione con il Physical Asset e, più in generale, quale sia il loro ruolo e obbiettivo come framework per la gestione dei Digital Twin nel contesto dell'IoT.

Successivamente, introdurremo un caso di studio (Capitolo 4), con l'obbiettivo non solo di confrontare i diversi metodi di sviluppo dei Digital Twin proposti dai due framework, ma anche di definire un approccio generale utilizzabile per lo sviluppo di un Digital Twin a partire dal relativo Physical Asset. Dopo l'implementazione del Physical Asset, condurremo un'analisi delle proprietà, eventi e azioni da mappare nel Digital Twin, definendo uno schema di comunicazione tra PA, DT e applicazioni esterne per fornire un quadro generale migliore su cui sviluppare i Digital Twin.

Infine, nel Capitolo 5, dopo aver analizzato i due framework e aver sviluppato i Digital Twin con essi, condurremo il confronto vero e proprio oggetto della tesi, evidenziando le differenze riscontrate nella gestione e nello sviluppo dei Digital Twin.

Capitolo 2

Analisi del framework Eclipse Ditto

Eclipse Ditto [**2**] è un framework open source utile per creare Digital Twin; è agnostico rispetto al dominio di progettazione e quindi può essere utilizzato in ambito industriale, residenziale, agricolo e in molti altri domini IoT.

L'obiettivo di Eclipse Ditto è fungere da middleware IoT, introducendo un livello di astrazione che semplifichi l'interazione delle soluzioni IoT con i dispositivi fisici, seguendo il pattern Digital Twin (Figura 2.1). In altre parole, Ditto assume le responsabilità tipiche di un "back-end" tradizionale: fornisce un'API per astrarre il sistema fisico, instrada le richieste dal sistema fisico alle applicazioni esterne, garantisce la disponibilità dei dati anche in caso di malfunzionamento del sistema fisico e notifica le parti interessate sugli eventi all'interno del dominio. Ditto offre agli utenti e alle aziende un modello generale su cui modellare i propri sistemi fisici, consentendo loro di evitare l'adozione di soluzioni proprietarie per la gestione e l'esposizione dei dati del proprio sistema fisico al mondo esterno.

Il framework fornisce una propria definizione di Digital Twin per eliminare le ambiguità presenti nel termine a causa delle diverse interpretazioni attribuitegli. Per Eclipse Ditto, un **Digital Twin** è un concetto per astrarre un dispositivo o una risorsa del mondo reale nella sua rappresentazione digitale, comprendendone tutte le capacità ed aspetti. Un Digital Twin quindi:

- Riflette un dispositivo o una risorsa fisica,
- Funge da "unica fonte di verità" per una risorsa fisica,
- Offre vari aspetti e servizi legati al dispositivo,
- Mantiene sincronizzati mondo digitale e fisico,

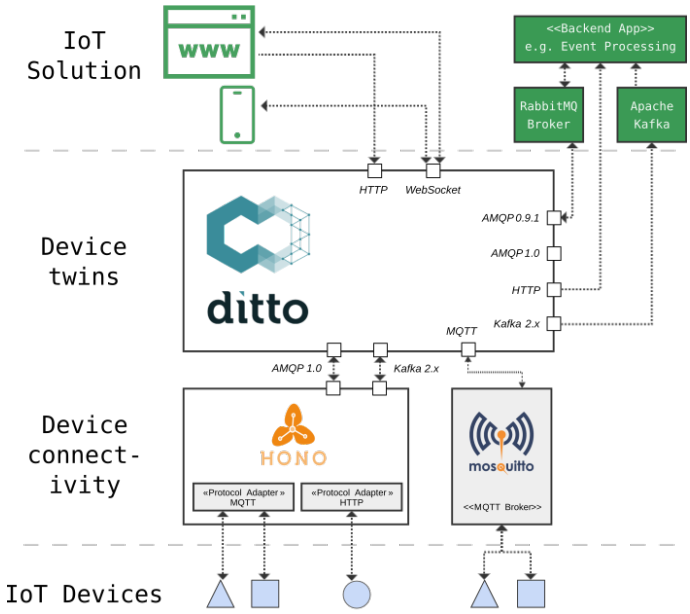


Figura 2.1: Interazione fra Ditto, dispositivi IoT e soluzioni IoT

- Può essere utilizzato sia in ambito industriale che in ambito consumatore.

Eclipse Ditto, come framework per i Digital Twin, fornisce gli strumenti necessari per interagire con essi, non solo come singoli elementi, ma anche come una popolazione e assicura che solo le parti autorizzate possano accedervi.

Dopo aver presentato una panoramica su Ditto, focalizzandosi sugli obbiettivi del framework e sulla sua interpretazione del paradigma dei Digital Twin, il presente capitolo si concentrerà sull'analisi dettagliata della struttura di Ditto. Inizieremo analizzando l'architettura di Ditto, definendo i servizi che consentono la gestione dei Digital Twin e le modalità di comunicazione di tali servizi. Successivamente, esploreremo i meccanismi impiegati da Ditto per modellare i Physical Asset nei rispettivi Digital Twin, analizzando anche le dinamiche di comunicazione tra il sistema fisico e quello digitale. Infine, affronteremo le API fornite da Ditto per interagire con i Digital Twin e il protocollo proprietario utilizzato per la trasmissione di messaggi in entrata e in uscita.

2.1 Architettura

Ditto è costituito da una serie di micro-servizi interconnessi, come evidenziato nella Figura 2.2. Ciascun micro-servizio:

- Dispone di un proprio archivio dati, accessibile e scrivibile esclusivamente da esso,
- Espone un'API composta da segnali (comandi, risposte ai comandi ed eventi),
- Può interagire con altri servizi solo attraverso segnali predefiniti.

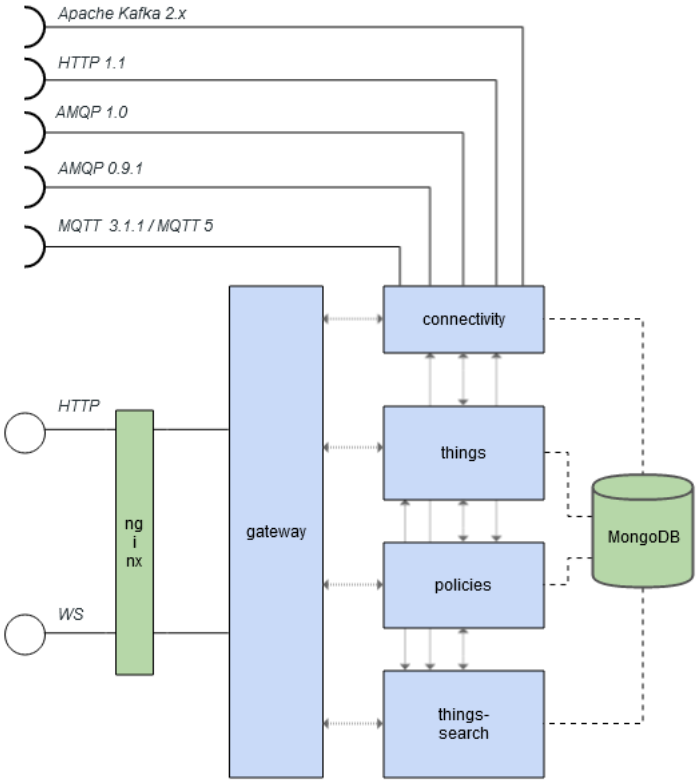


Figura 2.2: Servizi Ditto (in blu), dipendenze esterne (MongoDB e nginx) ed endpoint API forniti ed utilizzati esternamente

Tutti i micro-servizi operano all'interno dello stesso cluster Pekko [9]. Nella comunicazione, ciascun micro-servizio agisce come un server, esponendo endpoint TCP, ai quali gli altri micro-servizi possono inviare dati. I micro-servizi svolgono i seguenti compiti:

- **Policies**: gestisce la persistenza e l'applicazione (autorizzazione) delle Policy (vedi Sezione 2.2.3).
- **Things**: si occupa della persistenza e dell'applicazione (autorizzazione) delle Thing (vedi Sezione 2.2.1) e delle Feature (vedi Sezione 2.2.2).

- **Thing-Search**: monitora le modifiche apportate a Thing, Feature e Policy; aggiorna un indice di ricerca ottimizzato ed esegue query su quest'ultimo.
- **Gateway**: fornisce le API HTTP e WebSocket.
- **Connectivity**: gestisce la persistenza delle Connection (vedi Sezione 2.6), invia messaggi in Ditto Protocol (vedi Sezione 2.4) a broker esterni e riceve messaggi da essi.

2.2 Entità del Modello

Al fine di creare un Digital Twin, Ditto mette a disposizione diverse entità utilizzate per modellarlo. Per istanziare un Digital Twin, sarà necessario definire almeno: una **Policy** (Sezione 2.2.3), che determina le regole di accesso al Digital Twin, una **Thing** (Sezione 2.2.1) che specifica la struttura del Digital Twin e un insieme di **Feature** (Sezione 2.2.2) che modellano le proprietà e le capacità del Digital Twin.

La presente sezione introdurrà dettagliatamente tali entità. Successivamente, le Sezioni 2.5 e 2.6 analizzeranno le entità utilizzate da Ditto per stabilire il collegamento tra Digital Twin e Physical Asset.

2.2.1 Thing

Le **Thing** sono le entità fondamentali del modello utilizzate per modellare una risorsa fisica nel rispettivo Digital Twin. Una Thing potrebbe essere: un dispositivo fisico, un dispositivo virtuale, un'entità transazionale o qualsiasi altra risorsa che possa essere adeguatamente modellata. Ogni Thing è identificata da un Thing ID univoco.

L'accesso a una Thing è regolato da una Policy, identificata da un `policyId`, che definisce i soggetti autenticati con permessi di lettura (READ) e/o scrittura (WRITE) sulla Thing o le sue parti.

Ciascuna Thing incorpora una definizione associata ad un modello che ne delinea le capacità e le proprietà. Tale definizione può essere utilizzata anche per la ricerca di una Thing. Una definizione potrebbe, ad esempio, essere un URL contenente la descrizione di una Thing definita dal Web of Things (WoT) Thing Model (vedi Sezione 2.7.2). Per definire la struttura di una Thing, vengono utilizzati due elementi:

- **Attribute**: gestisce i meta-dati statici che non cambiano frequentemente,
- **Feature**: gestisce i dati di stato di una Thing.

2.2.2 Feature

Le **Feature** sono entità utilizzate per modellare tutte le funzionalità e i dati di una Thing, raggruppabili in uno specifico contesto tecnico. Per modellare i diversi contesti e aspetti di una Thing, possono essere utilizzate diverse Feature, tutte appartenenti alla stessa Thing e che non esistono al di fuori di essa (Figura 2.3). Come le Thing, anche le Feature sono identificate da una stringa univoca chiamata Feature ID.

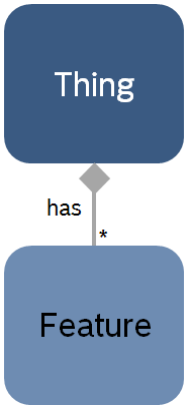


Figura 2.3: Una Thing può contenere varie Feature

I dati all'interno di una Feature sono gestiti coma una lista di proprietà. Ogni proprietà può essere un valore scalare o un oggetto complesso. In entrambi i casi, l'insieme delle proprietà è rappresentato da un unico oggetto JSON. Allo stesso modo delle proprietà, vengono gestite anche le "desired properties", utilizzate per rappresentare il valore atteso di una data proprietà.

Come per le Thing, Ditto permette di specificare una definizione per le Feature in modo da definire come queste siano strutturate e quale sia il loro comportamento e le loro capacità. Una definizione può essere vista come un "tipo" per una Feature; nonostante ciò, Ditto non assicura che una proprietà o una desired property appartengano al tipo specificato, spetta a chi invia i valori delle proprietà effettuare questo controllo.

2.2.3 Policy

Le **Policy** consentono allo sviluppatore di configurare l'accesso alle Thing e ad altre entità. Una Policy fornisce a qualcuno, chiamato **subject**, i permessi di lettura e/o scrittura di una data risorsa.

Il subject ID, `<subject-issuer>:<subject>`, contiene un prefisso che ne definisce l'emittente e il subject vero e proprio, separati da due punti. Ditto supporta diversi modi per autenticarsi in qualità di subject: nginx, google

(JSON Web Token), provider personalizzati conformi a OpenID Connect o utilizzando un altro provider di pre-autenticazione che utilizzi l'header HTTP `x-ditto-pre-authenticated`.

Una Policy permette di controllare l'accesso di diverse risorse:

- **Policy**: se viene concesso il permesso di scrittura alla root della Policy, l'utente sarà in grado di modificarla.
- **Thing**: l'accesso può essere consentito a tutta la risorsa o a una sua sottoparte.
- **Message**: i permessi saranno applicati a tutti i messaggi inviati da o verso la Thing a cui si riferisce la Policy. Per inviare Message ad una Thing è necessario un permesso di scrittura, mentre per ricevere i Message da una Thing è necessario un permesso di lettura.

2.3 Signal

I **Signal** sono uno dei concetti fondamentali di Ditto, principalmente utilizzati per la comunicazione interna al cluster Ditto. I Signal racchiudono le seguenti funzionalità:

- **Command**: richieste inviate al dominio per modificare (Modify Command) o recuperare (Query Command) informazioni relative a un Digital Twin gestito da Ditto.
- **Command Response**: risposta ad un Command che include informazioni sulla richiesta, in caso di successo, o un errore in caso contrario.
- **Error Response**: errore riportato al mittente del Command o del Message che lo ha causato.
- **Event**: evento che indica che è avvenuto qualcosa all'interno del Digital Twin, che non può essere annullato (irreversibile). Gli Event sono pubblicati all'interno del cluster Ditto per consentire ai servizi back-end di gestirli. Vengono anche notificati alle parti interessate ed autorizzate tramite WebSocket API, HTTP Server Sent Events o Change Notification (vedi Sezione 2.3.1).
- **Announcement**: a differenza degli eventi, gli Announcement anticipano che qualcosa sta per accadere nel Digital Twin. Ad esempio, prima che un Event sia creato e pubblicato, un Announcement potrebbe segnalare che l'evento si verificherà a breve.

Pattern di comunicazione Analizzando l'interazione tra un Physical Asset, le applicazioni esterne e Ditto, è possibile delineare un pattern di comunicazione dei Signal attraverso il diagramma di sequenza illustrato nella Figura 2.4. Questo modello di comunicazione può essere sintetizzato nei seguenti passaggi:

1. Ditto riceve un Command inviato da un PA e lo processa,
2. Una Success Response o una Error Response vengono restituite al mittente a seconda dell'esito del comando,
3. Inoltre, in caso di successo, un Event viene registrato nel database e pubblicato. L'evento descrive i cambiamenti applicati all'entità. Le parti interessate potranno poi iscriversi a tali eventi per monitorare l'evoluzione dello stato dell'entità.

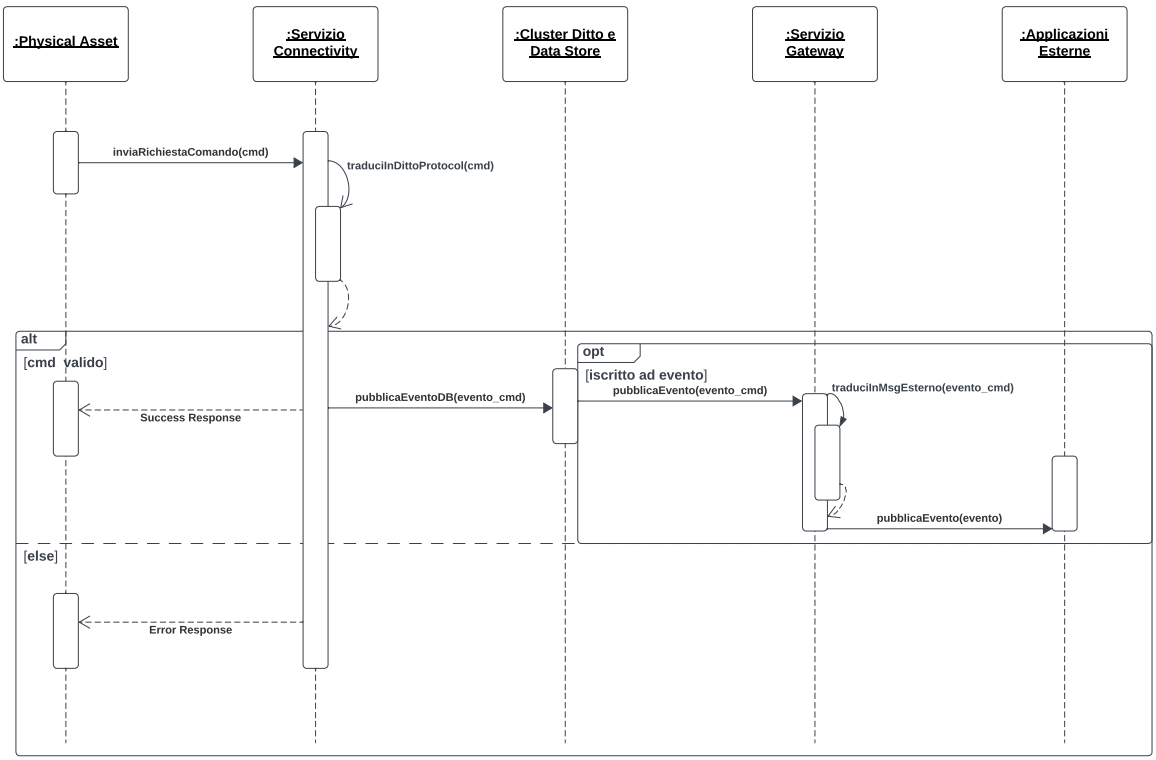


Figura 2.4: Pattern di comunicazione dei Ditto Signal

Questo processo illustra la sequenza di interazioni che avviene internamente a Ditto quando un comando viene ricevuto, elaborato e portato a termine. La

registrazione degli eventi nel database fornisce una traccia persistente delle modifiche, offrendo alle parti interessate un meccanismo per rimanere informate sullo stato corrente e sulle dinamiche dell'entità coinvolta.

2.3.1 Change Notification

Una **Change Notification** rappresenta un evento generato quando un'entità viene modificata. Questo meccanismo svolge un ruolo cruciale nel mantenere costantemente aggiornate le applicazioni esterne riguardo allo stato del Digital Twin. Nel contesto di Ditto, sono disponibili diversi metodi per ricevere notifiche relative a tali eventi:

- Attraverso l'API WebSocket , ogni client WebSocket riceve tutti gli eventi che il subject è autorizzato a ricevere sotto forma di messaggi Ditto Protocol.
- Utilizzando HTTP SSE, un consumatore di Server Sent Events (`EventSource`) è in grado di ricevere direttamente tutti gli eventi autorizzati al subject nel formato dell'entità modificata.
- Sfruttando le connessioni stabilite tramite il servizio di connectivity.

Gli eventi possono essere filtrati da parte di Ditto prima della trasmissione, mediante l'impiego di filtri basati sul namespace o su espressioni RQL (Resource Query Language).

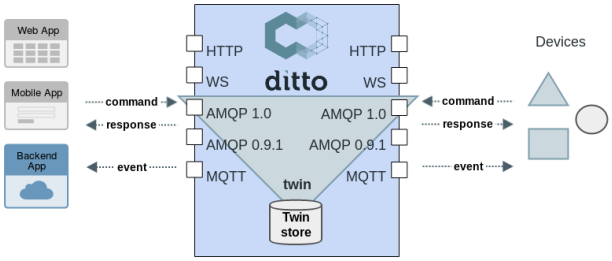
2.4 Ditto Protocol

Ditto introduce un protocollo di testo proprietario basato su JSON, denominato **Ditto Protocol**, per facilitare la comunicazione tra il Digital Twin e il dispositivo fisico. Questo protocollo definisce un insieme di comandi che possono essere elaborati sia dal dispositivo che dal Digital Twin.

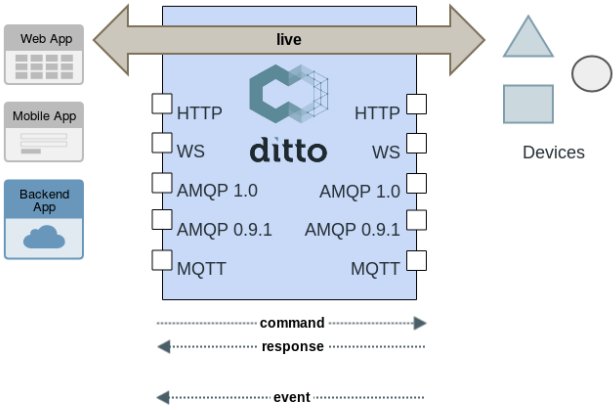
Il Ditto Protocol utilizza due canali di comunicazione per gestire aspetti distinti della comunicazione tra il Digital Twin e il dispositivo:

- Il canale **Twin** (Figura 2.5a) è legato alla rappresentazione digitale della Thing. Attraverso questo canale, è possibile leggere e modificare lo stato e le proprietà della Thing.
- Il canale **Live** (Figura 2.5b) instrada i comandi e i messaggi verso il dispositivo fisico. La gestione e l'esecuzione dei comandi ricevuti sono di competenza esterna a Ditto, il quale si occupa esclusivamente di verificare che solo le parti autorizzate possano inviare messaggi.

Solo i comandi legati alle Policy non specificano alcun canale nel Ditto Protocol.



(a) Canale Twin



(b) Canale Live

Figura 2.5: Canali del Ditto Protocol

2.4.1 Specifiche del protocollo

Per essere conforme al Ditto Protocol, un messaggio deve contenere:

- Ditto Protocol Envelope: dipende implicitamente dal protocollo utilizzato per il trasporto del messaggio. Descrive il contenuto del messaggio e consente al messaggio di essere instradato correttamente.
- Ditto Protocol Payload: contiene i dati del messaggio.

Il Protocol Envelope deve contenere i seguenti campi:

- `topic`: utilizzato per identificare l'entità destinataria, definire il tipo di canale e specificare il tipo di messaggio inviato,

- `headers`: header aggiuntivi in formato JSON,
- `path`: contiene il percorso per identificare l'entità a cui applicare il campo value,
- `fields`: i campi che devono essere inseriti nella risposta,
- `value`: il valore da applicare al percorso specificato,
- `extra`: permette di specificare i campi extra da inserire nel messaggio,
- `revision`: per gli eventi, contiene il numero di revisione dell'evento,
- `timestamp`: per gli eventi, contiene il timestamp di modifica dell'evento.

2.4.2 Protocol Topic

Il Ditto Protocol definisce un Topic che per ogni messaggio ha la seguente struttura: `{namespace}/{entity-name}/{group}/{channel}/{criterion}/{action}`:

- `namespace`: il namespace all'interno del quale si trova l'entità,
- `entity-name`: il nome dell'entità a cui è indirizzato il messaggio,
- `group`: il gruppo a cui appartiene l'entità. Possibili valori: things o policies a seconda che sia indirizzato ad una Thing o ad una Policy,
- `channel`: il tipo di canale utilizzato per il messaggio, da omettere nel caso delle Policy,
- `criterion`: specifica il tipo di azione contenuta nel messaggio. Possibili valori: commands, events, search, messages, errors, acks e announcements,
- `action`: campo opzionale, per comandi, eventi e messaggi che permette di specificare ulteriormente lo scopo del messaggio.

2.5 Message

I **Message** rappresentano un meccanismo utilizzato da Ditto per la trasmissione di informazioni da e verso il dispositivo fisico, facendo uso di un argomento arbitrario. I messaggi inviati al dispositivo consistono in operazioni che devono innescare un'azione sul sistema, mentre quelli provenienti dal

dispositivo rappresentano eventi o allarmi generati dal sistema stesso. In questo contesto, i messaggi non influenzano direttamente lo stato del Digital Twin o del dispositivo fisico, ma richiedono l'elaborazione dei dati ricevuti.

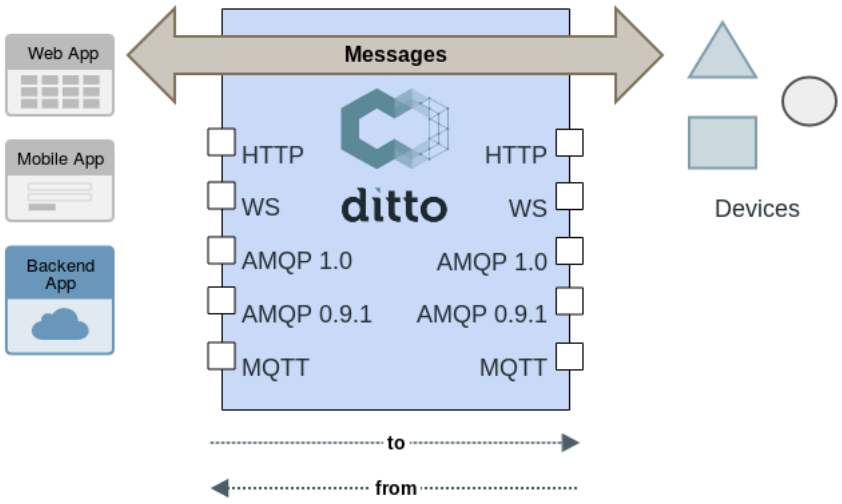


Figura 2.6: Ditto come router di Message

Il procedimento di instradamento dei messaggi, illustrato nella Figura 2.6, è gestito da Ditto in diverse fasi. Ditto procede all'accettazione dei messaggi attraverso due API (HTTP e Ditto Protocol), esegue poi una verifica per identificare la presenza di parti interessate attualmente connesse ed in grado di ricevere il messaggio ed infine attua l'effettivo instradamento dei messaggi e delle relative risposte tra i client connessi.

Ditto non implementa alcun servizio di conservazione dei messaggi; pertanto, nel caso in cui il dispositivo destinatario non sia attualmente connesso, il messaggio non sarà ricevuto.

I messaggi di Ditto sono costituiti da tre elementi obbligatori:

- **Direction**: da o verso,
- **Thing ID**: l'id della Thing destinataria o mittente del messaggio,
- **Subject**: il subject o topic del messaggio.

Inoltre, possono essere presenti ulteriori campi:

- **Feature ID**: se il messaggio fa riferimento ad una specifica Feature di una Thing,

- **content-type**: definisce il tipo di contenuto del payload del messaggio,
- **correlation-id**: necessario se Ditto dovesse rispondere al mittente del messaggio iniziale.

Per ricevere un messaggio per una Thing, è necessario possedere i permessi di lettura (READ) per quella Thing. Mentre, per inviare messaggi da una Thing, occorre disporre dei permessi di scrittura (WRITE) per quella Thing.

API

I messaggi possono essere inviati a Ditto attraverso diverse modalità:

- WebSocket API come messaggi in formato Ditto Protocol,
- HTTP API, sia come messaggi "fire-and-forget" che in modo bloccante quando si attende una risposta,
- Source Connection gestite da Ditto, le quali ricevono i messaggi inviati dal mittente in Ditto Protocol.

Mentre, i dispositivi e le applicazioni esterne possono ricevere messaggi attraverso:

- WebSocket API come messaggi in formato Ditto Protocol,
- Server Sent Events API,
- Target Connection gestite da Ditto.

2.6 Connection

Una **Connection** costituisce un canale di comunicazione dedicato alla trasmissione di messaggi tra un servizio e Ditto. In altri termini, una connessione rappresenta il meccanismo che consente ai Digital Twin creati con Ditto di interagire sia con il rispettivo Physical Asset, sia con le applicazioni esterne che ne richiedono i servizi. Una connessione richiede l'utilizzo di un protocollo di trasporto specifico per poter trasmettere i messaggi in Ditto Protocol. Ditto offre supporto sia per la comunicazione mono-direzionale che per quella bidirezionale. Tutte le connessioni sono gestite in modo centralizzato dal servizio di Connectivity.

Una connessione è composta da una serie di campi, di seguito riportiamo quelli più rilevanti:

- `id`: identificatore univoco della connessione,
- `name`: nome che descrive la connessione,
- `connectionType`: definisce il protocollo di trasporto utilizzato dalla connessione,
- `connectionStatus`: indica lo stato attuale della connessione,
- `uri`: l'URI che definisce l'endpoint remoto della connessione,
- `sources`: le sorgenti a cui la connessione si iscrive,
- `targets`: le destinazioni di pubblicazione della connessione.

L'obbiettivo di Ditto è delineare una struttura per le connessioni il più universale possibile, permettendo l'utilizzo di protocolli con differenti payload. Attualmente, sono supportati i seguenti tipi di connessione: AMQP, MQTT, HTTP e Kafka.

Il formato di `sources` e `targets` dipende dal `connectionType`.

2.6.1 Source

Le Source sono utilizzate per connettersi ai sistemi esterni o ai broker di messaggi, consentendo la ricezione di messaggi da tali fonti. Una Source include:

- Diversi indirizzi, interpretatati in modo diverso a seconda del tipo di connessione,
- Un consumer count che identifica quanti consumer collegare a ciascun indirizzo sorgente,
- Un authorization context che definisce quale subject viene utilizzato per autorizzare i messaggi provenienti dalla sorgente,
- Informazioni per filtrare i messaggi elaborati dalla sorgente,
- Un header mapper che consente di mappare gli header dei messaggi in header interni,
- Un reply-target, configurabile per gestire eventuali risposte ai messaggi in ingresso.

2.6.2 Target

I Target sono utilizzati per connettersi ai sistemi esterni o ai broker di messaggi, consentendo l'invio di messaggi a tali destinazioni. Un Target comprende:

- Un indirizzo interpretato in modo diverso a seconda del tipo di connessione,
- I topic di Ditto inviati al destinatario,
- Un authorization context che definisce il subject utilizzato per autorizzare i messaggi da parte del destinatario,
- Un header mapper che consente di mappare gli header in Ditto Protocol in header esterni.

2.6.3 Payload Mapping

Affinché Ditto possa elaborare i messaggi, è necessario che essi siano conformi al formato Ditto Protocol JSON. Dovendo supportare una varietà di dispositivi dotati di diversi protocolli di comunicazione, Ditto è dotato di una serie di "message mapper" in grado di convertire il payload di un messaggio esterno nel formato Ditto Protocol, e viceversa.

In particolare citiamo il JavaScript Mapping Engine, che consente di definire degli script JavaScript personalizzati per mappare qualsiasi tipo di payload in messaggi nel formato Ditto Protocol JSON. Ditto fornisce una serie di funzioni per semplificare il mapping utilizzando JavaScript:

- `buildDittoProtocolMessage()`: che permette di creare un messaggio in Ditto Protocol dati dei parametri in input,
- `buildTopic()`: che permette di creare un topic di Ditto dati dei parametri in input,
- `buildExternalMsg()`: che permette di creare dei messaggi esterni dati dei parametri in input.

2.7 Compatibilità con W3C Web of Things

Come già introdotto in precedenza, Ditto permette di specificare una definizione per una Thing o una Feature, documentando la struttura di queste entità e le capacità/comportamenti ad esse associate.

Per la definizione di una Thing/Feature, Ditto oltre a supportare Eclipse Vorto (progetto attualmente non più mantenuto attivamente e quindi escluso dalla nostra analisi), consente l'utilizzo di un URL HTTP che faccia riferimento a un Thing Model in formato JSON-LD, conforme alle specifiche del W3C Web of Things (WoT).

Nelle prossime sezioni, esamineremo il modello proposto dal W3C Web of Things (WoT) per poi valutare l'integrazione offerta da Ditto per tale modello.

2.7.1 W3C Web of Things

Il W3C Web of Things (WoT) Working Group [17] fornisce una serie di "building-blocks" volti a semplificare lo sviluppo di soluzioni IoT e a contrastarne la frammentazione utilizzando il paradigma Web. Vengono introdotti i concetti di:

- **Thing Description [14]**: utilizzata per descrivere esattamente un'istanza di un dispositivo o risorsa fisica. Oltre a riportare le proprietà del dispositivo, le azioni invocabili o gli eventi emessi, contiene gli endpoint effettivi delle API (tramite il campo `forms`) che permettono di interagire con il dispositivo.

- **Thing Model [15]**: definisce un modello generale, un'interfaccia, che una Thing Description può implementare. Tale modello si concentra quindi sulle possibili interazioni del dispositivo (proprietà, azioni ed eventi) e sulla loro semantica.

2.7.2 Integrazione con Eclipse Ditto

L'introduzione del Thing Model, rende il WoT una soluzione ideale per descrivere le capacità di un Digital Twin gestito da Ditto. Con la versione 2.4.0 di Ditto viene aggiunto il supporto al WoT; i Digital Twin possono essere collegati a un Thing Model, da cui Ditto può creare una Thing Description contenente le API del twin.

Tra i benefici derivanti dall'adozione del Thing Model in Ditto, si evidenziano:

- Possibilità di definire un modello per i dati (attributi e feature di Ditto), specificando il tipo di dato, possibili restrizioni, valori di default e unità di misura.

- Possibilità di definire un modello per i messaggi, indicando possibili input/output o situazioni di errore.

- Capacità di fornire un contesto semantico (campo `@context` di JSON-LD), facendo riferimento ad ontologie esistenti,
- Interoperabilità: il modello che rappresenta il dispositivo è uno standard riutilizzabile anche per altre soluzioni e piattaforme che supportando il WoT,
- Introspezione dei Digital Twin che possono descriversi autonomamente.

Concetti del WoT mappati in Ditto

Il mapping di una Thing Description in una Ditto Thing può essere effettuato a diversi livelli di complessità; quello più completo vede una Thing Description descrivere una Ditto Thing (tabella 2.1) e le Feature di una Thing a loro volta descritte ognuna dalla propria Thing Description (tabella 2.2). Il Thing Model viene poi referenziato nel campo `definition` di una Thing o una Feature, da cui l'integrazione WoT di Ditto è in grado di generare una Thing Description.

Concetto WoT	**Concetto Ditto**
Thing	Thing
Properties	Attributes
Actions	Message diretto verso Thing
Events	Message inviato da Thing
Composizione tramite `tm:submodel`	Features

Tabella 2.1: Concetti Thing Model mappati in Ditto Thing

Concetto WoT	**Concetto Ditto**
Thing	Feature
Properties	Feature Properties
Actions	Message diretto verso Feature
Events	Message inviato da Feature

Tabella 2.2: Concetti Thing Model mappati in Ditto Feature

Funzionalità dell'integrazione

L'integrazione WoT per Ditto offre diverse funzionalità: referenzia l'URL del Thing Model nella definizione di Thing/Feature, può generare una Thing Description per una Thing/Feature referenziata da un Thing Model e in fase

di creazione di una nuova Thing, può generare uno scheletro JSON a partire dall'URL di un Thing Model accessibile pubblicamente.

Trattiamo in dettaglio la funzionalità di generazione dello scheletro JSON, che verrà utilizzata nelle fasi successive di sviluppo dei DT (vedi Sezione 4.2.2):

- Ditto verifica che la `definition` contenga un URL HTTP valido,
- Scarica l'URL referenziato e verifica che si tratti di un Thing Model valido,
- Salva il Thing Model nella cache locale di Ditto,
- Utilizza il Thing Model come template per generare oggetti JSON:
  - Attribute generati a partire dalle `properties` del Thing Model,
  - Feature generate a partire dai `tm:sumbmodel` utilizzando `instanceName` come Feature ID,
  - Proprietà delle Feature generate a partire dalle `properties` dei sotto-modelli.
- Se il campo `default` viene specificato per una proprietà, questo verrà utilizzato come valore, altrimenti viene utilizzato il "valore neutro" del tipo di dati specificato,
- In caso di errore durante la creazione dello scheletro, viene creata una Thing senza lo schema, contenente solo il riferimento al Thing Model in `definition`.

2.8 API

Ditto mette a disposizione due modalità di interazione:

- Una Rest-like HTTP API che permette di creare, leggere, modificare e cancellare le Thing e i relativi dati,
- Una JSON-based WebSocket API che implementa il Ditto Protocol.

Le due API sono pressoché equamente potenti e consentono di eseguire le stesse operazioni sui dati delle Thing, oltre a inviare e ricevere messaggi. L'HTTP API può essere utilizzata, ad esempio, in dispositivi meno potenti che non dispongono di un runtime Java o per lo sviluppo di interfacce utente basate sul Web. L'API WebSocket risulta vantaggiosa per raccogliere flussi di dati consistenti da un broker di messaggi, per monitorare i dispositivi in tempo reale, per applicazioni Web basate su eventi, ecc.

Altri aspetti in cui le due interfacce differiscono:

- Scambio di messaggi: le richieste WebSocket sono non bloccanti con risposte asincrone, mentre le richieste HTTP sono bloccanti con risposte sincrone,
- Sicurezza: WSS - WebSocket over Transport Layer Security contro HTTPS.
- Connessione: WebSocket è connection-oriented con una connessione persistente per ridurre la latenza e aumentare il throughput, mentre HTTP connectionless con una minore allocazione di risorse.

Capitolo 3

Analisi del framework White Label Digital Twin

La libreria Java **White Label Digital Twin** (**WLDT**) è stata progettata per offrire supporto nella progettazione, nello sviluppo e nell'implementazione di Digital Twin all'interno di ecosistemi IoT. La libreria è basata sulle più recenti definizioni di Digital Twin provenienti dal contesto industriale e scientifico, WLDT concepisce i Digital Twin come componenti software attivi.

La libreria fa riferimento alla definizione di Digital Twin, già illustrata nella Sezione 1.2 e ripresa in [10]. La risorsa del mondo reale, definita come Physical Asset (PA), si riferisce invece ad un'entità che ha una manifestazione o una rilevanza nel mondo fisico, oltre a possedere un ciclo di vita ben definito.

WLDT mira a massimizzare la *modularità*, la *riusabilità* e la *flessibilità* per modellare efficacemente le risorse fisiche intelligenti nelle loro controparti digitali. La libreria cerca di semplificare la progettazione e l'implementazione di DT, fornendo un insieme di funzionalità di base che consentano l'adozione dei DT nelle applicazioni IoT.

Un'istanza WLDT è un'entità software general-purpose che implementa tutte le caratteristiche e le funzionalità di un DT in esecuzione sul cloud o in un edge. La sua caratteristica distintiva è la genericità, grazie alla quale è in grado di associarsi a qualsiasi oggetto fisico per crearne una replica digitale ed estenderne le funzionalità.

I requisiti che hanno guidato la progettazione e lo sviluppo della libreria includono:

- **Semplicità**: gli sviluppatori devono essere in grado di creare facilmente una nuova istanza utilizzando i moduli esistenti o personalizzandoli a seconda delle proprie esigenze applicative.

- **Estensibilità**: l'API deve essere facilmente estensibile per permettere ai programmatori di personalizzare la configurazione o aggiungere nuove funzionalità eseguendo diversi moduli contemporaneamente.

- **Portabilità e Disponibilità per i micro-servizi**: un DT implementato con WLDT deve essere eseguibile su qualsiasi piattaforma senza modifiche o personalizzazioni. L'obbiettivo è fornire una base semplice e leggera con un insieme strategico di funzionalità orientate all'IoT, consentendo agli sviluppatori di creare facilmente applicazioni DT modellate come agenti software indipendenti e confezionate come micro-servizi.

Dopo aver fornito una visone generale su WLDT, delineando gli obbiettivi che la libreria persegue nella creazione e gestione dei Digital Twin, il presente capitolo si concentrerà su un'analisi approfondita di tutti gli aspetti di WLDT. Inizieremo esaminando come WLDT modella i Digital Twin, concependoli come entità dinamiche composte da una serie di componenti e caratterizzate da diversi stati che ne definiscono il comportamento e l'interazione con il Physical Asset. Successivamente, approfondiremo l'architettura della libreria, illustrando i componenti che consentono di istanziare un Digital Twin e facilitano la sua interazione con il Physical Asset e le applicazioni esterne.

3.1 Astrazione e Modello Digital Twin

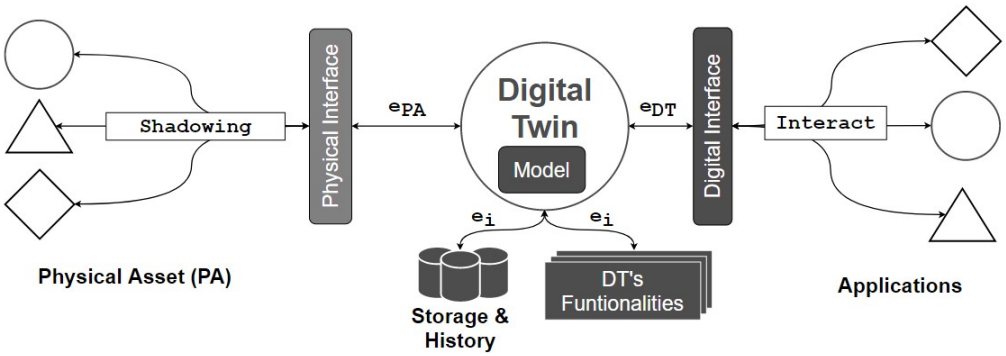


Figura 3.1: Componenti principali di un Digital Twin astratto

Un Digital Twin rappresenta quindi un ponte tra il mondo digitale e quello fisico. Come illustrato nella figura 3.1, i componenti fondamentali di un Digital Twin astratto (successivamente mappati all'interno della libreria WLDT) sono:

- **Physical Interface**: entità responsabile del processo iniziale di *digitalizzazione* o *shadowing* e di mantenere sincronizzati PA e DT durante il

ciclo di vita. L'interfaccia può eseguire vari **Physical Asset Adapter** per interagire con il PA e digitalizzare gli eventi fisici, in base alla sua natura e ai protocolli e formati dati supportati.

- **Digital Interface**: componente che gestisce le variazioni e gli eventi interni al DT attraverso entità digitali esterne e consumatori. Utilizza diversi **Digital Adapter** per gestire eventi e interazioni digitali, garantendo l'interoperabilità con le applicazioni esterne.

- **Modello del DT**: il modulo definisce il comportamento del DT e le sue funzionalità aumentate. Supporta l'esecuzione di vari moduli configurabili e riutilizzabili e funzionalità che possono gestire eventi fisici e digitali in base al comportamento implementato. Il modello è responsabile della gestione e dell'aggiornamento del **Digital Twin State**.

Il modello del Digital Twin consente di catturare e rappresentare il PA a un livello di astrazione adeguato, ignorando gli aspetti irrilevanti e modellando solo le informazioni a livello di dominio anziché gli aspetti tecnologici.

Il collegamento tra Physical e Digital Twin è definito come **Shadowing**, un processo continuo e (quasi) in tempo reale di aggiornamento dello stato interno del DT in relazione ai cambiamenti del PA.

Ciascun Digital Twin è dotato di un modello interno che definisce come il corrispondente PA è modellato a livello digitale. La rappresentazione del DT, denotata come **Digital Twin State** è definita in termini di:

- **Proprietà**: rappresentano gli attributi osservabili del PA come dati etichettati i cui valori possono cambiare dinamicamente nel tempo, a seconda delle evoluzione dello stato del PA.

- **Eventi**: rappresentano gli eventi a livello di dominio che possono essere osservati nel PA.

- **Relazioni**: rappresentano i collegamenti tra il PA modellato e altre risorse fisiche appartenenti all'organizzazione, tramite il collegamento dei corrispondenti DT. Possono essere osservate, create e modificate nel tempo, ma non fanno parte dello stato del PA, bensì del suo contesto operativo.

- **Azioni**: rappresentano le azioni che possono essere invocate nel PA tramite l'iterazione con il DT o direttamente nel DT nel caso in cui non siano disponibili nel PA; in tal caso, il DT sta ampliando le capacità del PA.

Una volta definito il modello, possiamo delineare lo stato dinamico del DT come la combinazione di proprietà, eventi, relazioni e azioni associati ad un

timestamp che indica l'istante di sincronizzazione tra la risorsa fisica e la controparte digitale.

3.2 Processo di Shadowing

Il processo di **Shadowing** consente di mantenere sincronizzato lo stato del Digital Twin (Digital Twin State) con quello della corrispondente risorsa fisica, come definito nel modello del DT.

In particolare, ogni modifica significativa dello stato del Physical Asset viene gestita attraverso tre passaggi principali:

1. Ogni cambiamento rilevante nello stato della risorsa fisica è modellato in un `physical_event` (`e_pa`),
2. L'evento viene propagato al Digital Twin,
3. Dato l'evento `e_pa`, il Digital Twin viene aggiornato applicando una *shadowing function*, che dipende dal modello.

Il processo di shadowing consente anche di invocare possibili azioni nel Physical Asset. Il Digital Twin riceve una richiesta di azione (`digital_action`) nella sua interfaccia digitale, applica la shadowing function per convalidarla e, infine, la propaga all'interfaccia fisica. Una `digital_action` non modifica direttamente lo stato del Digital Twin, poiché ogni cambiamento può avvenire solo come risultato della shadowing function tra il PA e il DT.

3.3 Ciclo di vita del Digital Twin

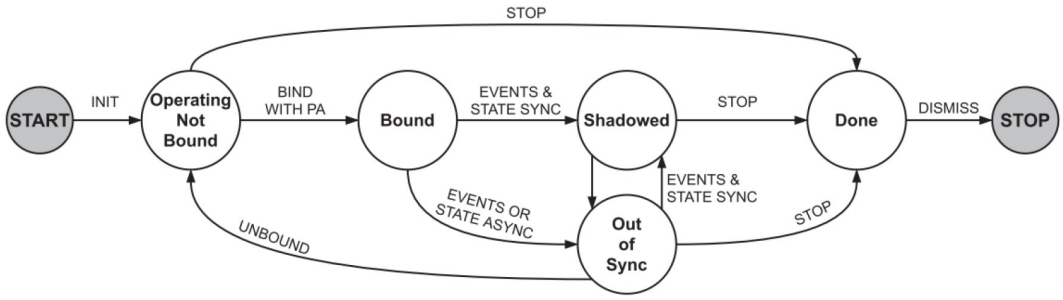


Figura 3.2: Ciclo di vita del Digital Twin

La modellazione del concetto di Digital Twin include la definizione del suo ciclo di vita. Basandosi sulla letteratura scientifica, la libreria WLDT modella

un ciclo di vita composto da 5 stati che il DT attraversa dalla sua inizializzazione fino alla disattivazione. La Figura 3.2 fornisce una rappresentazione grafica dei seguenti stati:

- **Operating & Not Bound**: in questo stato, il DT è stato inizializzato e tutti i suoi componenti sono attivi, ma non è ancora stato associato al corrispondente PA.
- **Bound**: stato in cui si trova il DT dopo che il processo di associazione è andato a buon fine. Il processo di associazione consente di collegare le due parti e abilita il flusso di eventi bidirezionale tra di loro.
- **Shadowed**: stadio raggiunto dal DT quando inizia il processo di shadowing e il suo stato viene correttamente sincronizzato con quello del PA.
- **Out of Sync**: questa fase determina la presenza di un errore nel processo di shadowing. Il DT non è in grado di gestire né gli eventi di allineamento dello stato né gli eventi generati dal livello applicativo.
- **Done**: stadio raggiunto quando il processo di shadowing è disattivato, ma il DT può comunque continuare a ricevere richieste da applicazioni esterne.

3.3.1 Da Unbound a Bound

Supponiamo che il PA utilizzi un protocollo P per comunicare e che sia connesso con il DT tramite la Physical Interface con un Adapter specifico per il protocollo P.

Per passare dallo stato Unbound allo stato Bound, il DT deve acquisire conoscenza della risorsa rispetto al protocollo P. Il processo di shadowing può avere inizio solo dopo l'associazione e l'ottenimento di una descrizione delle proprietà e delle capacità della controparte fisica.

L'Adapter P comunica con il PA attraverso il protocollo P e fornisce una **Physical Asset Description** (**PAD**). Una volta che il Physical Adapter è stato correttamente associato al PA e la PAD è stata generata, il DT può passare dallo stato Unbound allo stato Bound.

I concetti principali che caratterizzano la PAD sono:

- Viene utilizzata per descrivere la lista di proprietà, azioni e relazioni del PA.
- Ciascun Physical Adapter genera una PAD dedicata, associata alla sua prospettiva del PA e alle sue capacità di leggere dati ed eseguire azioni.

- Sarà compito del DT gestire multiple PAD per costruire la replica digitale del PA.

- La PAD verrà impiegata dal DT per gestire il processo di shadowing e per mantenere sincronizzati la replica digitale con la controparte fisica.

Il passaggio da uno stato Unbound a uno stato Bound è fondamentale per abilitare il DT a interagire con il PA e avviare il processo di shadowing, garantendo così la sincronizzazione tra la rappresentazione digitale e la risorsa fisica nel corso del loro ciclo di vita.

3.3.2 Da Bound a Shadowed

La libreria definisce la procedura necessaria per passare dallo stato Bound a quello Shadowed, in cui il Digital Twin ha identificato le capacità del PA tramite la PAD ricevuta ed è in grado di avviare il processo di sincronizzazione con il mondo fisico. I passi necessari sono i seguenti:

1. Il modello definisce le proprietà da monitorare nel PA ed inizia ad osservarle tramite gli adapter.

2. I Physical Adapter coinvolti comunicano con il PA, ricevono dati e generano eventi (`ePA`) che notificano cambiamenti delle proprietà fisiche.

3. Gli eventi ricevuti saranno utilizzati dal Digital Twin Model per eseguire la shadowing function e calcolare il nuovo stato del DT.

4. Il DT può passare dallo stato Bound a quello Shadowed fino a quando è in grado di mantenere una sincronizzazione appropriata con il PA nel corso del tempo e di generare e mantenere il Digital Twin State.

Il **Digital Twin State** è caratterizzato da una lista di proprietà, una lista di azioni e una lista di relazioni. Questi elementi possono essere sia presenti nel PA che generati dal DT Model come combinazione di proprietà, azioni e relazioni del PA. Il Digital Twin State è gestito tramite la Shadowing Function ed espone un insieme di metodi per essere manipolato; quando si verifica un cambiamento nel Digital Twin State un evento (`eDT`) sarà generato.

Ogni variazione ed evoluzione durante il ciclo di vita del DT genera una serie di eventi (`eDT`) che possono essere usati dalla Digital Interface e in particolare dai Digital Adapter per esporre lo stato del DT, le sue proprietà e capacità al mondo digitale esterno. Allo stesso tempo, gli eventi `eDT` possono essere utilizzati dai Digital Adapter per scatenare azioni che possono essere propagate al PA associato al DT.

3.4 Struttura della Libreria

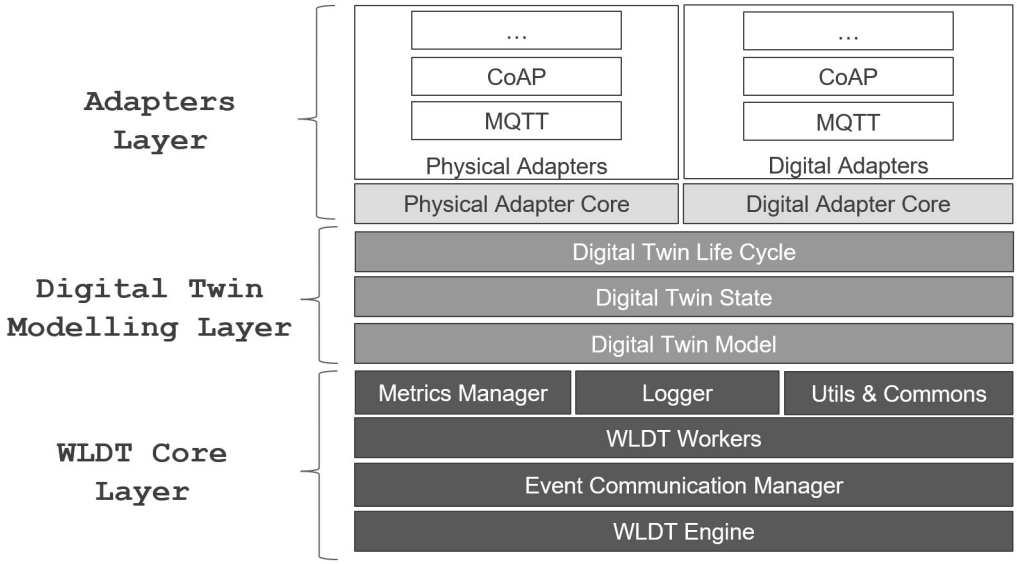


Figura 3.3: Struttura WLDT

La figura 3.3 presenta i componenti chiave dell'architettura di WLDT, attraverso i quali viene implementato un singolo Digital Twin. L'architettura è suddivisa in tre livelli distinti: il `core` della libreria, il livello di modellazione del DT e il livello degli adapter. Ogni componente principale ha le seguenti caratteristiche:

- **WLDT Engine**: definisce il motore multi-thread della libreria che consente l'esecuzione e il monitoraggio di più worker simultaneamente. Responsabile della gestione dei vari moduli interni dell'architettura, può essere considerato il nucleo stesso del DT.
- **WLDT Event Bus**: rappresenta il bus degli eventi interno, progettato per supportare la comunicazione tra i diversi componenti dell'istanza DT. Ciascun componente WLDT può pubblicare sul bus eventi condiviso e definire un filtro eventi per specificare a quali eventi è interessato.
- **WLDT Workers**: modella un componente interno generale che costituisce un'unità eseguibile dal WLDT Engine. Ad eccezione del Digital Twin State, ogni componente successivo rappresenta un'implementazione concreta di un WLDT Worker.
- **Digital Twin State**: struttura lo stato del DT definendo la lista di proprietà, eventi e azioni. Espone una serie di metodi per essere manipolato

dalla Shadowing Model Function. Ogni volta che il Digital Twin State viene modificato, genera il corrispondente evento DT per notificare tutti i componenti.

- **Shadowing Model Function**: il componente responsabile di definire il comportamento del DT interagendo con il Digital Twin State. Implementa il processo di shadowing che mantiene sincronizzati DT e PA. Basato su una specifica implementazione di un WLDT Worker, chiamata Model Engine, per essere eseguito dal WLDT Engine. La Shadowing Model Function è fondamentale e deve essere estesa dallo sviluppatore per concretizzare il modello del DT.

- **Physical Adapter**: definisce le funzionalità essenziali che l'estensione individuale, riferita a un protocollo, deve implementare.

- **Digital Adapter**: definisce l'insieme di callback che la specifica implementazione può utilizzare per essere notificata dei cambiamenti nello stato del DT.

Pertanto, per creare un DT utilizzando WLDT, è necessario definire una Shadowing Model Function e almeno un Physical Adapter e un Digital Adapter, per consentire la connessione con il PA e l'utilizzo del DT da parte di applicazioni esterne. Una volta implementati i tre componenti, è possibile istanziare un WLDT Engine e avviare il ciclo di vita del DT.

Capitolo 4

Sviluppo dei Digital Twin utilizzando i due framework

Il presente capitolo si concentra sullo sviluppo di Digital Twin attraverso l'utilizzo dei due framework precedentemente presentati, Eclipse Ditto (Capitolo 2) e WLDT (Capitolo 3). L'obiettivo principale è fornire un confronto dettagliato tra i diversi strumenti messi a disposizione degli sviluppatori per la creazione dei Digital Twin.

La prima sezione del capitolo (Sezione 4.1) inizia con lo sviluppo del Physical Asset, la cui rappresentazione digitale sarà successivamente modellata nei Digital Twin oggetto dell'analisi. Questo processo di sviluppo fornisce il contesto iniziale per comprendere le caratteristiche e le funzionalità dei Digital Twin che saranno successivamente implementati e confrontati.

4.1 Caso di studio: Smart Bridge

Per il nostro caso di studio, abbiamo selezionato uno **Smart Bridge** come Physical Asset, un ponte arricchito da sensori e attuatori che lo rendono idoneo all'interazione con Digital Twin e applicazioni IoT.

4.1.1 Specifiche

Il Physical Asset che intendiamo modellare è la simulazione di un Smart Bridge, un sistema montato sopra un ponte dotato di funzionalità smart. Le principali caratteristiche includono:

- **Monitoraggio del livello dell'acqua**: in caso di allarme, il sistema può aprire delle valvole per far defluire l'acqua in corsi secondari,

- **Sistema di illuminazione smart**: il ponte è dotato di un sistema di illuminazione intelligente che accende le luci in base alla presenza di persone che lo attraversano.

La figura 4.1 illustra i sensori e gli attuatori utilizzati per simulare il funzionamento del Physical Asset. Questi includono due LED verdi (G1 e G2), un LED rosso, un servo motore (HS-53), un sensore di prossimità ad ultrasuoni (Sonar HC-SR04), un sensore di movimento (Passive Infrared Sensor - PIR) e un sensore di luminosità (Light Dependent Resistor - LDR). Il tutto è collegato a un System-on-a-Chip ESP32-S3 che fornisce i servizi di connettività necessari per la comunicazione con i Digital Twin.

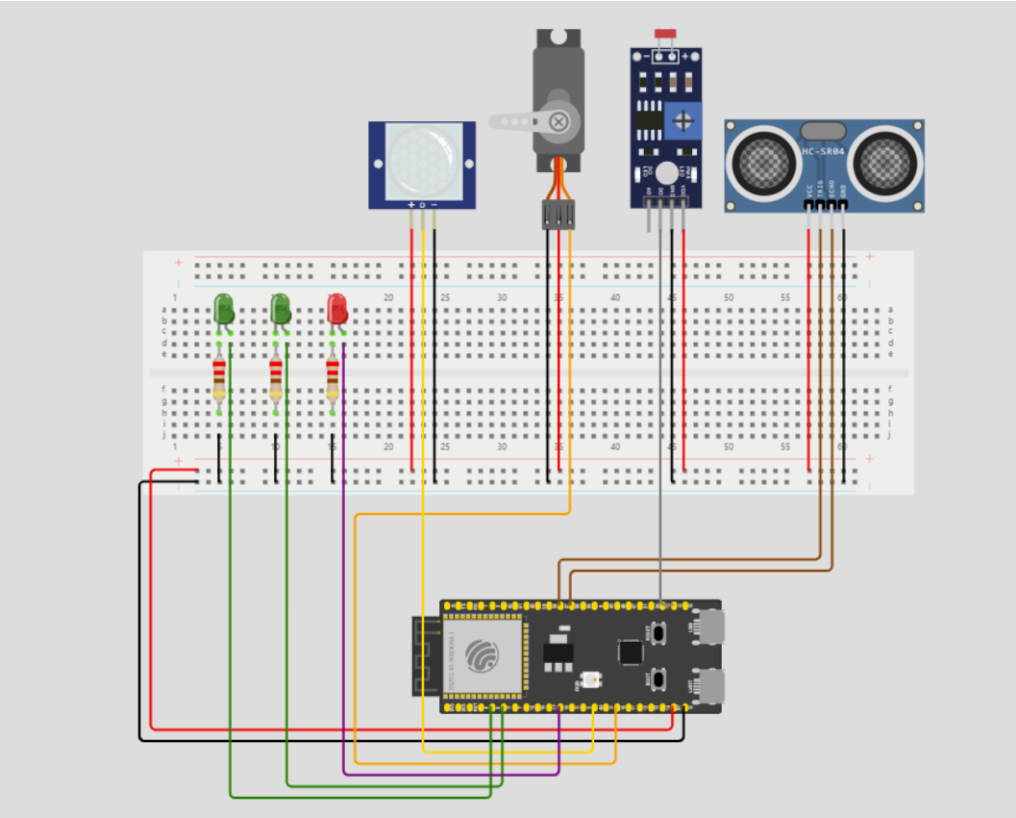


Figura 4.1: Circuito dello Smart Bridge

Il sottosistema di illuminazione smart comprende il PIR P, il sensore di luminosità (LS) e il LED G1. Il sottosistema opera come segue:

- Se qualcuno viene individuato sul ponte dal PIR, il LED G1 sarà accesso o spento in base al livello di luminosità rilevato da LS,
- Se il livello di luminosità è inferiore ad una certa soglia TH, allora il LED G1 sarà acceso, altrimenti sarà spento,

- Il LED G1 sarà spento dopo un certo periodo se nessuno viene individuato sul ponte o se il livello di luminosità supera una certa soglia TH.

Il sottosistema di monitoraggio del livello dell'acqua comprende il sonar S, il servo motore M, il LED G2 e il LED rosso. Il sottosistema opera come segue:

- Il sonar S monitora costantemente il livello dell'acqua,
- Il motore M gestisce l'apertura/chiusura di una valvola che consente all'acqua del fiume di defluire.
- Se il livello dell'acqua è sotto una soglia WL1, allora il sistema è in una situazione normale: il LED G2 è accesso e il LED rosso spento, ad indicare che il ponte può essere utilizzato.
- Se il livello dell'acqua è compreso tra WL1 e WL2 il sistema è in una situazione di pre-allarme: il LED rosso inizia a lampeggiare con un periodo di 2 secondi.
- Infine, se il livello dell'acqua è sopra la soglia WL2 il sistema è in una situazione di allarme:
  - Il sottosistema di illuminazione smart viene spento,
  - Il LED G2 è spento e il LED rosso accesso
  - La valvola è aperta di ALPHA gradi che dipendono linearmente dal livello dell'acqua. L'apertura della valvola cambia dinamicamente in base al livello attuale dell'acqua.
  - In questa situazione, un utente esterno può prendere il controllo della valvola impostando uno stato manuale tramite l'interfaccia utente (vedi Sezione 4.1.3 e Sezione 4.1.4).

4.1.2 Modello Digital Twin

Dopo aver esaminato il funzionamento dello Smart Bridge, identifichiamo le proprietà e le funzionalità che desideriamo siano modellate dai Digital Twin. Prima di procedere, è necessario delineare lo schema di comunicazione tra PA, DT e applicazioni esterne coinvolte.

Come evidenziato nella Figura 4.2, i due sottosistemi del sistema Smart Bridge sono stati implementati come due PA distinti ed indipendenti (consultare la Sezione 4.1.3 per ulteriori dettagli), che non comunicano direttamente

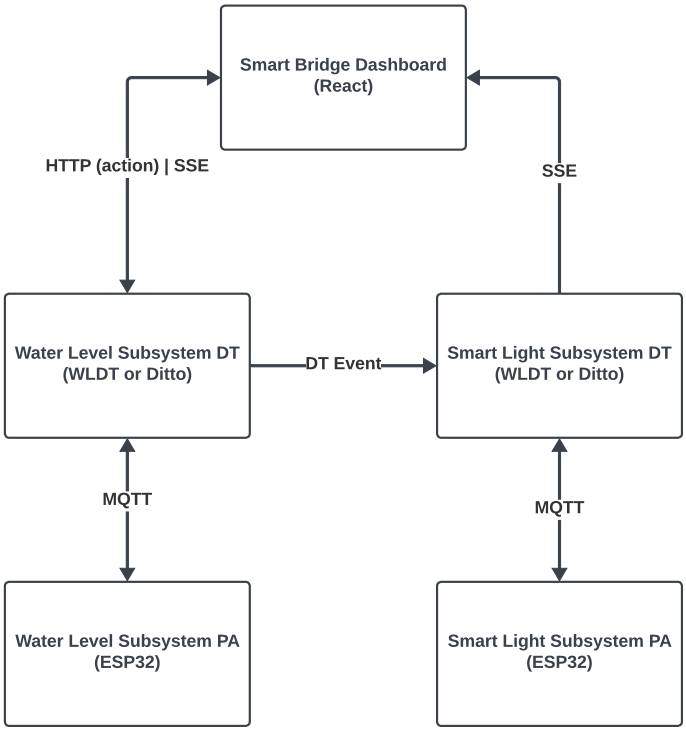


Figura 4.2: Schema di comunicazione tra PA, DT e applicazioni esterne

tra di loro, ma tramite i rispettivi DT. Questa scelta consente una struttura più semplice per i singoli PA e facilita il loro eventuale riutilizzo in altri progetti che non richiedano l'interazione diretta tra i due sottosistemi.

Successivamente, i due PA, che da ora in poi chiameremo **Water Level Subsystem** e **Smart Light Subsystem**, sono stati modellati all'interno dei rispettivi DT. In particolare, il DT del Water Level Subsystem è progettato per estendere le funzionalità del PA, generando un evento `alarm` quando il sistema si trova in una situazione di allarme. Questo evento consente ad altri sistemi in ascolto, come ad esempio il DT dello Smart Light Subsystem, di prendere decisioni adeguate. La comunicazione tra i due DT è gestita in modo diverso in base alle funzionalità offerte dai framework, come illustreremo in seguito. Mentre, la comunicazione tra DT e PA avviene attraverso il protocollo MQTT.

Infine, i due DT comunicano con un'interfaccia web (consultare la Sezione 4.1.4), che consente la consultazione dello stato, aggiornato tramite Server Sent Events, e, in situazioni di allarme, l'invio di un'azione `manual` per prendere il controllo del Water Level Subsystem. Anche in questo caso, il tipo di comunicazione dipende dal framework utilizzato.

Proprietà

I due Physical Asset presentano diverse proprietà suddivisibili in tre categorie: proprietà derivanti dai sensori, proprietà di stato e proprietà da applicare agli attuatori. Nonostante sia possibile calcolare tutte le altre proprietà dalle informazioni dei sensori, si è scelto di mantenere un set più ampio di proprietà, calcolate direttamente dai PA e poi trasmesse ai DT. Questa scelta consente a un utente esterno di verificare lo stato di tutti gli attuatori e sensori, e potenzialmente controllarne il corretto funzionamento. Di seguito è riportato l'elenco delle proprietà considerate per i due PA.

Proprietà del Water Level Subsystem

- `status`: indica lo stato corrente del sottosistema. Possibili valori: *normal*, *pre-alarm* o *alarm*.
- `green-led`: proprietà booleana che indica lo stato di accensione del led verde.
- `red-led`: proprietà composta da due sotto-proprietà booleane, `on` e `blinking`, che indicano rispettivamente lo stato di accensione del led rosso e se il led sta lampeggiando.
- `valve`: indica di quanti gradi è aperta la valvola (`int`).
- `distance`: indica la distanza dell'acqua dal ponte (`float`).
- `manual`: proprietà booleana che indica se è attivo il controllo manuale.

Proprietà dello Smart Light Subsystem

- `status`: indica lo stato corrente del sottosistema. Possibili valori: *sys_on* o *sys_off*.
- `dark`: proprietà booleana che indica se il ponte è buio.
- `detected`: proprietà booleana che indica se sono presenti persone sopra al ponte.
- `smart-light`: proprietà composta da due sotto-proprietà booleane, `on` e `waiting`, che indicano rispettivamente lo stato di accensione del led verde e se il led è acceso ma in attesa di essere spento.

La proprietà `status` dello Smart Light Subsystem è particolare in quanto non è modificata solo da variazioni derivanti dal PA (sys_on), ma viene aggiornata anche direttamente dal DT in risposta ad eventi provenienti dal Water Level Subsystem (sys_off). Sarà poi il DT a notificare il PA dell'aggiornamento della proprietà. In questo caso, possiamo affermare che il DT dello Smart Light Subsystem estende le funzionalità del PA.

Eventi

Nessuno dei due PA espone direttamente degli eventi osservabili. Tuttavia, come precedentemente illustrato, il Digital Twin del Water Level Subsystem genera un evento `alarm` ogni volta che la proprietà `status` ha valore *alarm*, estendendo così le funzionalità del PA per notificare eventuali applicazioni esterne in ascolto (ad esempio, lo Smart Light Subsystem).

Azioni

Il PA del Water Level Subsystem consente a un utente esterno di assumere il controllo della valvola attraverso un'azione denominata `manual`. Tale azione è in grado di ricevere due tipologie di messaggi:

- Un messaggio contenente il campo `manual`, che consente di impostare lo stato manuale nel sistema e di modificare la relativa proprietà `manual`.
- Un messaggio contenente il campo `angle`, che, se il sistema è in modalità manuale, permette di modificare l'angolo di apertura della valvola.

In questo contesto, il compito dei DT è quello di esporre questa azione alle applicazioni esterne, come ad esempio un'interfaccia utente dedicata alla gestione dei due sistemi (vedi Sezione 4.1.4). L'azione non è direttamente accessibile alle applicazioni esterne tramite il PA; al contrario, queste devono interfacciarsi con il DT corrispondente. Come descritto successivamente, a seconda del framework utilizzato, il DT indirizza l'azione verso il PA con la possibilità di eseguire controlli aggiuntivi sulla correttezza dei dati inviati.

4.1.3 Implementazione

La simulazione del Physical Asset dello Smart Bridge [13] è stata implementata utilizzando un SoC ESP32-S3, con C++ all'interno dell'ambiente di sviluppo PlatformIO. Disponendo di un unico SoC per il progetto, i due PA che costituiscono lo Smart Bridge sono stati implementati sullo stesso sistema. Tuttavia, sono concettualmente separabili e indipendenti, senza comunicazione diretta tra di loro nel SoC, ma solo tramite i Digital Twin esterni.

Durante lo sviluppo del sistema, le tre *Finite-State Machine* (Figura 4.3), individuate per descrivere il funzionamento del PA, sono state mappate in tre Task di FreeRTOS [3] (il sistema operativo utilizzato all'interno del SoC), oltre a un quarto Task aggiuntivo utilizzato per la comunicazione con i DT:

- **Water Level Task**: gestisce il sottosistema omonimo. Ad ogni esecuzione del task, legge il valore del sonar da cui ricava la distanza dall'acqua. In base a questo valore, verrà modificato lo stato del task e, di conseguenza, saranno impostati i valori degli attuatori.

- **Smart Light Task**: gestisce il sottosistema di illuminazione intelligente. Ad ogni esecuzione del task, se il sottosistema è accesso, vengono letti i valori del sensore di movimento e del sensore di luminosità per determinare lo stato di accensione della Smart Light (vedi dettaglio in Figura 4.4). Lo stato di funzionamento del sotto-sistema (accesso/spento) viene determinato ad ogni esecuzione del task tramite una variabile condivisa con il Communication Task, la quale viene impostata dal DT connesso. Di default il sistema è accesso, e verrà spento solo in caso il Water Level Subsystem si trovi in una situazione di allarme, seguendo lo schema di comunicazione illustrato precedentemente.

- **Blink Task**: task che si occupa di far lampeggiare il led rosso con un certo periodo quando il Water Level Subsystem è in una situazione di pre-allarme. In questo caso, facendo parte dello stesso sottosistema la comunicazione con il Water Level Task avviene internamente al SoC tramite variabili condivise.

- **Communication Task**: gestisce la comunicazione da e verso i DT. Utilizza il protocollo MQTT per pubblicare messaggi nei topic utilizzati per notificare i DT dei cambiamenti nelle proprietà e si iscrive ai topic da cui riceve aggiornamenti e azioni dai DT. La comunicazione avviene integralmente mediante messaggi in formato JSON. Inoltre, il Communication Task mantiene il valore pregresso dello stato di ogni task e delle sue proprietà, così i messaggi vengono inviati solo quando si verifica un cambiamento e non ad ogni esecuzione del task.

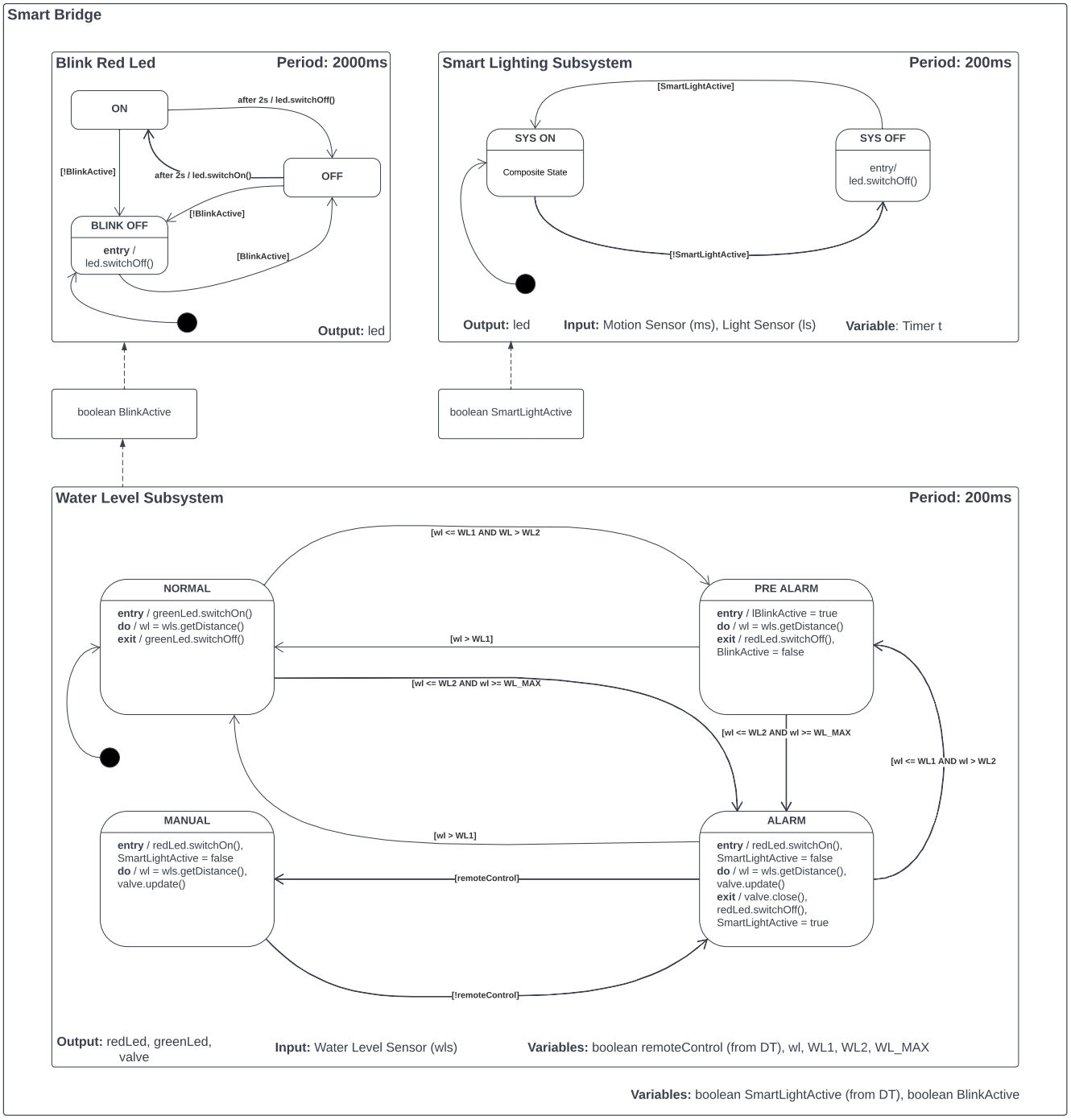


Figura 4.3: Diagramma delle Finite-State Machine utilizzate nello Smart Bridge

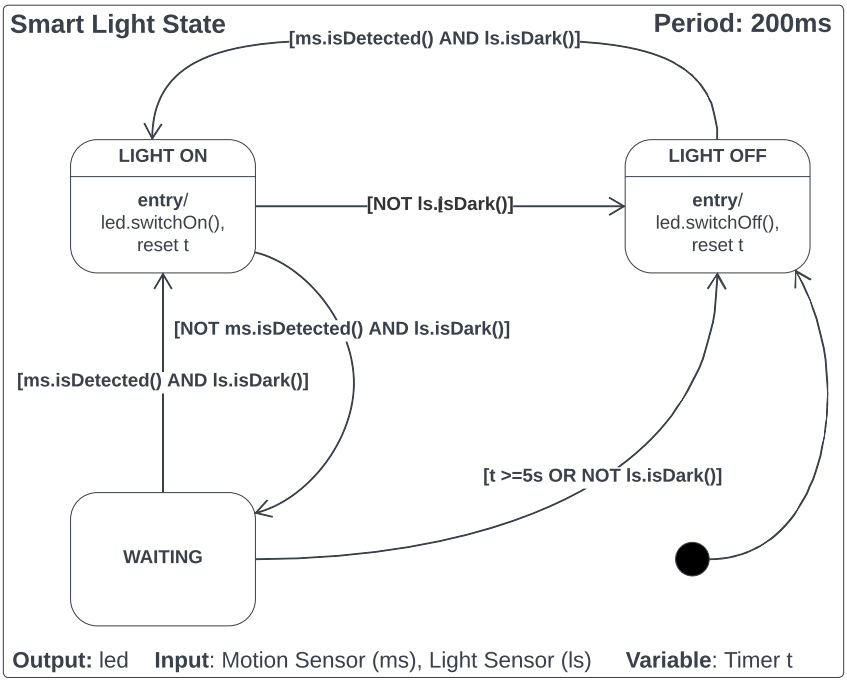


Figura 4.4: Dettaglio Finite-State Machine della Smart Light

4.1.4 Dashboard Web

La sezione di sviluppo comprende anche la realizzazione di un'interfaccia web [1], che permette di monitorare quasi in tempo reale lo stato dei PA (Figura 4.5) e di inviare loro richieste di azioni attraverso i rispettivi Digital Twin.

L'interfaccia, sviluppata utilizzando la libreria JavaScript ReactJS [12], è compatibile sia con il framework Eclipse Ditto che con WLDT. Questa flessibilità è ottenuta semplicemente configurando gli URL degli endpoint a cui effettuare le richieste.

L'interfaccia web interagisce con i DT in due modi principali:

- **Richieste HTTP POST**: vengono inviate richieste di azioni per assumere il controllo manuale del Water Level Subsystem (Figura 4.6).

- **Server Sent Events** (**SSE**): si istanziano due oggetti `EventSource` per aprire due connessione HTTP persistenti, attraverso le quali la dashboard riceve gli eventi di modifica delle proprietà da parte dei rispettivi DT.

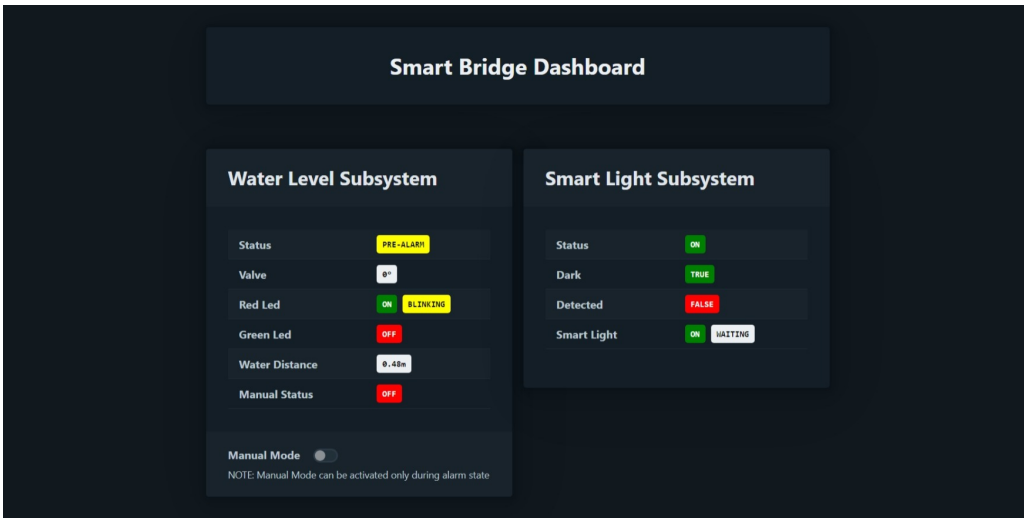


Figura 4.5: Dashboard con Water Level Subsystem in stato di pre-allarme

Le modalità di comunicazione dai DT verso la dashboard, implementate nei due framework, vengono dettagliate nelle successive sezioni (Sezione 4.2 e Sezione 4.3).

4.2 Sviluppo di Digital Twin utilizzando Eclipse Ditto

Per l'implementazione dei DT, Eclipse Ditto fornisce diverse client SDK, che altro non sono che implementazioni delle due API in diversi linguaggi di programmazione (ad esempio, Java e JavaScript). Date le dimensioni contenute del progetto, i DT sono stati creati utilizzando esclusivamente la HTTP API mediante comandi `curl`.

Per avviare Eclipse Ditto, è necessario disporre di un servizio MongoDB, delle immagini Docker dei micro-servizi di Ditto e di uno strumento come `docker-compose` per eseguire Ditto. Una volta avviato, sarà disponibile un'istanza di MongoDB come database di supporto (non direttamente parte di Ditto, ma necessario per il suo funzionamento), tutti i micro-servizi Ditto e un'istanza di `nginx` che funge da proxy inverso, eseguendo un'autenticazione di base in ascolto su una porta locale.

Per creare un DT utilizzando Ditto, è necessario definire almeno:

- Una Policy per regolare l'accesso alla Thing e alle sue parti;
- Una Thing con relativi Attribute e Feature, che modellino le proprietà e le funzionalità del PA;

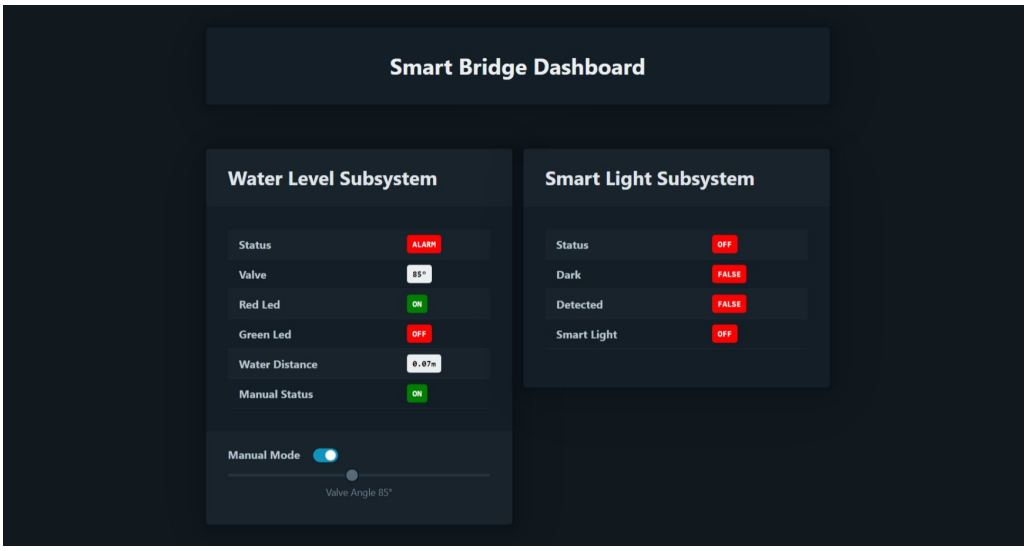


Figura 4.6: Dashboard con Water Level Subsystem in stato manuale

- Una o più Connection per sincronizzare il DT con il PA e, se necessario, esporre le sue proprietà al mondo esterno.

4.2.1 Policy

Poiché la sicurezza non è il focus principale del progetto, è stata creata una Policy predefinita composta da due voci:

- `owner`: con soggetto `nginx:ditto`, che consente l'accesso in lettura e scrittura a tutte le risorse.
- `connection` con soggetto `connection:mqtt`, utilizzabile per le connessioni, consente l'accesso in lettura/scrittura a tutte le entità, ad eccezione delle Policy stesse.

Una volta definita la configurazione della Policy, è possibile crearla utilizzando il comando `curl` attraverso l'API HTTP di Ditto, come mostrato nel Listato 4.1.

```
curl -X PUT 'http://localhost:8080/api/2/policies/namespace:policyID'
-u 'ditto:ditto'
-H 'Content-Type: application/json'
-d '<policy-config>'
```

Listato 4.1: Creazione di una Ditto Policy

4.2.2 Thing

Per definire la struttura delle due Ditto Thing destinate a mappare i due sottosistemi, è stata utilizzata l'integrazione di Ditto con le WoT Thing Description. Questa integrazione consente di creare uno scheletro delle proprietà e delle funzionalità del PA mediante un WoT Thing Model.

I Thing Model utilizzano il formato **JSON-LD** (JSON for Linked Data) [6] per rappresentare le informazioni. JSON-LD è un formato di interscambio di Linked Data, progettato intorno al concetto di "contesto" che collega le proprietà di un oggetto in un documento JSON ai concetti di un'ontologia. Nel caso del WoT Thing Description il `@context` ha il valore `https://www.w3.org/2022/wot/td/v1.1` per identificare il documento come una Thing Description 1.1.

Altri campi utilizzati per definire un Thing Model includono:

- `@type`: una keyword di JSON-LD per associare un tipo a un oggetto, nel caso del Thing Model, ha il valore `tm:ThingModel`.
- `title`: campo obbligatorio che specifica un titolo interpretabile dall'uomo.
- `description`: un campo che fornisce una descrizione aggiuntiva dell'oggetto.
- `version`: una campo che definisce la versione del modello.
- `properties`: una lista delle proprietà dell'oggetto, ognuna caratterizzata dai campi `title`, `description`, `type`, `default` (opzionale) ed altri campi facoltativi che possono essere utilizzati per definire unità di misura, valore massimo, valore minimo, ecc.
- `links`: un campo che consente di definire relazioni di composizione (`tm:submodel`) o di estensione (`tm:extends`) con altri Thing Model.

Nell'implementazione del progetto, all'interno del Thing Model sono state mappate solo le proprietà dei PA, mentre eventi e azioni sono stati modellati manualmente tramite Connection e Message.

Per ogni proprietà mappata in una Ditto Feature, è stato creato un Thing Model che ne definisce la struttura ed il tipo (vedi esempio nel Listato 4.2).

```
{
    "@context": [
        "https://www.w3.org/2022/wot/td/v1.1",
        {
            "@language" : "en"
```

```
        }
    ],
    "@type" : "tm:ThingModel",
    "id" : "water-level-status",
    "title" : "Water Level Status",
    "description" : "Status of the Water Level Subsystem",
    "version": {
        "model": "1.0.0"
    },
    "properties": {
        "status": {
            "title": "Status",
            "description" : "Current Water Level Subsystem Status",
            "type": "string",
            "enum" : ["NORMAL", "PRE_ALARM", "ALARM"],
            "default" : "NORMAL"
        }
    }
}
```

Listato 4.2: Proprietà `status` del Water Level Subsystem definita tramite Thing Model

Successivamente, è stato creato un Thing Model per ognuna delle due Ditto Thing destinate a mappare i PA (vedi esempio nel Listato 4.3). Le Feature sono importate tramite la chiave `links`, specificando per ognuna una voce con i seguenti campi:

- `rel`: con valore `tm:submodel`.
- `href`: contenete l'URL del Thing Model della Feature a cui ci si riferisce.
- `type`: con valore `application/tm+json`.
- `instanceName`: indicante il nome da attribuire alla feature.

```
{
    "@context": [
        "https://www.w3.org/2022/wot/td/v1.1",
        {
            "@language" : "en"
        }
    ],
    "@type": "tm:ThingModel",
    "id" : "smart-light-subsystem",
```

```
    "title": "Smart Light Subsystem",
    "description": "The smart light subsystem is composed by a
       motion sensor, a light sensor and a smart light",
    "version": {
        "model": "1.0.0"
    },
    "links": [
        {
            "rel": "tm:submodel",
            "href": "https://raw.githubusercontent.com/Fab-Ver/
            SmartBridgeDitto/main/wot_models/components/
            motion-sensor-1.0.0.tm.jsonld",
            "type": "application/tm+json",
            "instanceName": "motion-sensor"
        },
        {
            "rel": "tm:submodel",
            "href": "https://raw.githubusercontent.com/Fab-Ver/
            SmartBridgeDitto/main/wot_models/components/
            light-sensor-1.0.0.tm.jsonld",
            "type": "application/tm+json",
            "instanceName": "light-sensor"
        },
        {
            "rel": "tm:submodel",
            "href": "https://raw.githubusercontent.com/Fab-Ver/
            SmartBridgeDitto/main/wot_models/components/
            smart-light-status-1.0.0.tm.jsonld",
            "type": "application/tm+json",
            "instanceName": "status"
        },
        {
            "rel": "tm:submodel",
            "href": "https://raw.githubusercontent.com/Fab-Ver/
            SmartBridgeDitto/main/wot_models/components/
            smart-light-1.0.0.tm.jsonld",
            "type": "application/tm+json",
            "instanceName": "smart-light"
        }
    ]
}
```

Listato 4.3: Thing Model del Water Level Subsystem

Dopo aver creato i Thing Model (disponibili in un URL pubblico), per generare le due Ditto Thing è possibile utilizzare il comando `curl` sfruttando l'API HTTP di Ditto, come mostrato nel Listato 4.4.

```
curl --location --request PUT -u ditto:ditto
    'http://localhost:8080/api/2/things/namespace:thingID'
--header 'Content-Type: application/json'
--data-raw '{"policyId": "ditto.default:policy",
"definition": "<urlThingModel>"}'
```

Listato 4.4: Creazione di una Ditto Thing sfruttando l'integrazione WoT

4.2.3 Connection

Una volta creati i Digital Twin, è essenziale stabilire una comunicazione tra essi, i PA e il mondo esterno. Per il progetto, è stato definito il seguente schema di comunicazione:

- I due DT comunicano con i rispettivi PA tramite due connessioni MQTT.
- Gli eventi `alarm` del Water Level Subsystem vengono inviati allo Smart Light Subsystem tramite messaggi in Ditto Protocol (Modify-Command).
- Il Water Level Subsystem riceve le richieste di azioni dalla Dashboard tramite i messaggi Ditto (richiesta POST con adeguata autorizzazione) e li inoltra al proprio PA tramite una terza connessione MQTT dedicata esclusivamente all'indirizzamento dei messaggi.
- Lo Smart Light Subsystem notifica il proprio PA di una variazione nella proprietà `status` dovuta al Water Level Subsystem attraverso una quarta connessione MQTT dedicata solo agli eventi prodotti in seguito alle variazioni della proprietà.
- Per aggiornare la Dashboard tramite Server Sent Events inviati dai DT, è sufficiente che questa crei un `EvenSource` per ogni DT verso l'endpoint `/things/<thingId>` per ricevere le *change notification* della Thing specificata (sezione 2.3.1).

Vediamo ora come impostare i campi di una Connection per utilizzarla con il protocollo MQTT:

- `connectionType`: valore `mqtt`.
- `connectionStatus`: valore `open`.

- `uri`: specificare il broker a cui connettersi.
- `sources`: indica una lista di topic MQTT da cui ricevere i messaggi, la QoS (Quality-of-Service) desiderata e l'`authorizationContext`.
- `targets`: indica un topic MQTT su cui pubblicare i messaggi derivanti dai topic Ditto specificati nel campo `topics`.

Per il progetto, sono state create le seguenti Connection:

- Due connessioni per comunicare con i PA, in cui i `targets` sono vuoti mentre le `sources` includono i topic MQTT in cui sono ricevuti gli aggiornamenti da parte del PA. Per entrambe le connessioni è stata poi definita una funzione di mapping utilizzando il mapper JavaScript, che a seconda del topic su cui viene inviato il messaggio, crea dei comandi in Ditto Protocol per aggiornare i valori del DT. La funzione di mapping relativa al Water Level Subssytem è anche responsabile di inviare i comandi che riguardando gli eventi `alarm` allo Smart Light Subsystem (vedi esempio nel Listato 4.5).
- Una connessione per far comunicare direttamente la Dashboard e il PA tramite messaggi Ditto, in cui le `sources` sono vuote, mentre i `targets` indicano l'indirizzo in cui pubblicare il messaggio e specificano nei `topics` il valore `_/_/things/live/messages` per indicare che i messaggi devono essere inviati su questa connessione (vedi esempio nel Listato 4.6).
- Una connessione tra PA e DT del Water Level Subsystem che specifica solo i `targets` e come valore di `topics` indica che gli eventi di cambiamento dello stato devono essere indirizzati sulla connessione.

```
{
    "name": "WLS-MQTT Connection",
    "connectionType": "mqtt",
    "connectionStatus": "open",
    "failoverEnabled" : true,
    "uri": "tcp://test.mosquitto.org:1883",
    "sources": [{
        "addresses":
            ["subsystems/org.eclipse.ditto:water-level-subsystem/#"],
        "authorizationContext": ["connection:mqtt"],
        "qos": 0
    }],
    "targets": [],
    "mappingContext": {
```

```
        "mappingEngine": "JavaScript",
        "options": {
            "incomingScript": "function
                mapToDittoProtocolMsg(headers, textPayload,
                bytePayload, contentType) {...}"
        }
    }
}
```

Listato 4.5: Connessione con PA del Water Level Subsystem con mapping dei messaggi

```
{
    "name": "Messages Connection",
    "connectionType": "mqtt",
    "connectionStatus": "open",
    "failoverEnabled" : true,
    "uri": "tcp://test.mosquitto.org:1883",
    "sources": [],
    "targets": [{
        "address": "subsystems/messages/{{ thing:id }}",
        "topics" : [
            "_/_/things/live/messages"
        ],
        "authorizationContext": ["connection:mqtt"],
        "qos": 0
    }]
}
```

Listato 4.6: Connessione con PA del Water Level Subsystem per re-indirizzare i messaggi

Dopo aver definito la configurazione delle varie connessioni, queste possono essere create con il comando curl, tramite l'API HTTP di Ditto, come illustrato nel Listato 4.7.

```
curl -X 'POST' 'http://localhost:8080/api/2/connections'
-H 'accept: application/json'
-H 'Content-Type: application/json'
-u 'devops:foobar'
-d '<connection-config>'
```

Listato 4.7: Creazione di una Ditto Connection

4.3 Sviluppo di Digital Twin utilizzando WLDT

Come precedentemente illustrato, per creare un Digital Twin utilizzando WLDT è necessario definire una Shadowing Function e almeno un Physical Adapter ed un Digital Adapter. Nelle prossime sezioni, illustreremo le scelte effettuate per l'implementazione di ognuna di queste componenti. WLDT è sviluppato in Java, pertanto tutto il codice prodotto per la creazione dei DT è stato realizzato all'interno di un progetto Java.

4.3.1 Physical Adapter

Lo sviluppatore può utilizzare Physical Adapter esistenti o crearne dei nuovi. Nel caso del progetto, non è stato creato nessun nuovo Physical Adapter poiché la libreria ne fornisce già per il protocollo MQTT [8], utilizzato per la comunicazione con il PA. Illustreremo comunque i passaggi necessari alla creazione di un Physical Adapter, per poi spiegare le scelte effettuate con l'adapter MQTT.

Implementare un Physical Adapter

In generale, un Physical Adapter estende la classe `PhysicalAdapter`, responsabile della comunicazione con il mondo fisico, gestisce i seguenti compiti:

- Genera un PAD che descrive le proprietà, gli eventi, le azioni e le relazioni disponibili nel PA utilizzando la classe `PhysicalAssetDescription`.
- Genera eventi fisici utilizzando la classe `PhysicalAssetEventWldtEvent` associati alla variazione di qualsiasi aspetto dello stato fisico.
- Gestisce le richieste di azioni provenienti dal mondo digitale, tramite la Shadowing Function, implementando il metodo `onIncomingPhysical Action` che processa gli eventi di tipo `PhysicalAssetEventWldtEvent`.

Per creare un nuovo Physical Adapter è quindi necessario implementare i seguenti metodi:

- `onAdapterStart`: metodo callback utilizzato per notificare che l'adapter è stato correttamente inizializzato nel ciclo di vita del DT.
- `onAdapterStop`: metodo callback invocato quando l'adapter viene fermato e dismesso dal core.
- `onIncomingPhysicalAction`: metodo callback chiamato quando un evento viene inviato dalla Shadowing Function in seguito alla ricezione di un azione tramite un Digital Adapter.

Il primo passo, all'interno del metodo `onAdapterStart`, è generare e pubblicare una `PhysicalAssetDescription` che descriva le capacità e le caratteristiche della risorsa, permettendo alla Shadowing Function di decidere come digitalizzarla. Il PAD può essere generato manualmente secondo la natura della risorsa connessa o generato automaticamente da una "scoperta" o da una configurazione passata all'adapter. Dopo avere generato il PAD è necessario notificare la sua creazione alla Shadowing Function tramite il metodo `notifyPhysicalAdapterBound(pad)`.

Utilizzo del MQTT Physical Adapter nel progetto

Una volta che l'adapter MQTT è connesso al broker desiderato, permette di definire proprietà ed eventi da includere nel modello, associati al topic di ingresso a cui l'adapter si iscriverà per ricevere gli aggiornamenti. Per le azioni, è necessario specificare un topic in uscita nel quale l'adapter pubblicherà l'azione inviata dai Digital Adapter. In tutti i casi, è possibile definire un funzione che mappi i messaggi in ingresso nel valore corrispondente per aggiornare proprietà/eventi, o i messaggi in uscita nel valore desiderato per la richiesta.

Per il Water Level Subsystem, ogni proprietà del sistema è stata mappata in una corrispondente proprietà di WLDT, mentre l'azione `manual` è stata associata a un'azione WLDT. Successivamente, è stato creato l'evento `alarm` che estende le funzionalità del PA; ogni volta che viene ricevuto un aggiornamento sul topic di stato, il sistema verifica lo stato e genera un evento `alarm true/false`, che sarà poi propagato ai Digital Adapter. Questi, a loro volta, lo esporranno alle applicazioni esterne in ascolto.

Anche per lo Smart Light Subsystem, ogni proprietà del sistema è stata mappata in una corrispondente proprietà di WLDT, mentre il PA non espone né eventi né azioni. Contrariamente a tutte le altre proprietà, che sono aggiornate a seconda di variazioni nel PA, la proprietà di stato viene aggiornata mettendosi in ascolto dell'evento `alarm` prodotto dal Digital Adapter del Water Level Subsystem, che determina lo stato di accensione/spegnimento del sistema. In questo caso, quindi il DT sta estendendo le funzionalità del PA.

4.3.2 Shadowing Function

Dopo aver definito il Physical Adapter, procediamo con la definizione del nucleo del Digital Twin tramite la definizione della Shadowing Function. Quest'ultima si occupa di:

- Gestire la PAD ricevuta dal Physical Adapter per individuare le proprietà, gli eventi, le relazioni e le azioni disponibili nel PA che saranno mappate all'interno del Digital Twin State.

- Gestire notifiche e callback associate alla variazione di una proprietà fisica o alla generazione di un evento fisico.

- Processare le richieste di azione provenienti dal mondo digitale, che devono essere validate e inoltrate al Physical Adapter appropriato.

La Shadowing Function ha la responsabilità di costruire e mantenere aggiornato lo stato del DT. Per fare ciò, utilizza un'interfaccia chiamata `IDigitalTwinState`, implementata nella classe `DefaultDigitalTwinState`. La Shadowing Function ha accesso diretto (in lettura e scrittura) al Digital Twin State attraverso la variabile `digitalTwinState`. Questa classe espone una serie di metodi per creare, leggere, aggiornare ed eliminare proprietà, azioni, eventi e relazioni all'interno del Digital Twin.

Per implementare una Shadowing Function, è possibile estendere la classe base `ShadowingModelFunction`, che fornisce una serie di callback chiamate in seguito agli aggiornamenti nel Physical o Digital Adapter. Sia per il Water Level Subsystem che per lo Smart Light Subsystem, sono state sviluppate due Shadowing Function che, oltre a gestire lo stato e mettersi in ascolto per modifiche, instradano le variazioni di proprietà, eventi e relazioni dai Physical Adapter ai Digital Adapter e viceversa per le azioni.

Ecco una breve illustrazione dei metodi implementati:

- `onDigitalTwinBound(Map<String, PhysicalAssetDescription> map)`: questo metodo viene chiamato quando il DT passa dallo stato Unbound allo stato Bound. Al suo interno, vengono analizzate tutte le proprietà, le azioni, le relazioni e gli eventi del PAD. Per ognuno di essi, viene creata la corrispondente entità nel Digital Twin State. Dopo la registrazione di un'entità, si può iniziare ad osservarla per ricevere una notifica in caso di variazioni. Ad esempio, il metodo per osservare una proprietà è `observePhysicalAssetProperty(property)`. Dopo aver registrato tutte le entità, è necessario chiamare il metodo `notifyShadowingSync()` per notificare al DT Core che la fase di associazione è stata completata con successo e che il DT ha valutato il suo stato interno in base a quanto definito dai Physical Adapter.

- `onPhysicalAssetPropertyVariation`: questo metodo viene chiamato quando viene rilevata una variazione di una proprietà fisica nel Physical Adapter. Il metodo riceve un evento di tipo `PhysicalAssetProperty WldtEvent<?>` contenente tutte le informazioni sull'aggiornamento della proprietà. Nell'implementazione realizzata, il nuovo valore viene semplicemente mappato nella proprietà corrispondente tramite il metodo `this.digitalTwinState.updateProperty`.

- `onPhysicalAssetEventNotification`: questo metodo viene chiamato quando viene ricevuto un evento fisico da un Physical Adapter. La funzione riceve un parametro `PhysicalAssetEventWldtEvent<?>` contenete tutte le informazioni sull'evento generato. Nell'implementazione realizzata l'evento viene semplicemente ridirezionato verso i Digital Adapter in ascolto tramite il metodo `his.digitalTwinState.notifyDigitalTwinStateEvent`, che crea una nuova istanza di `DigitalTwinStateEventNotification` contenente tutte le informazioni sull'evento generato.

- `onDigitalActionEvent`: questo metodo viene chiamato dal Digital Adapter quando una richiesta di azione viene ricevuta nella Digital Interface. La funzione riceve un parametro `DigitalActionWldtEvent<?> digitalActionWldtEvent` che descrive l'azione ricevuta. Anche in questo caso l'azione viene semplicemente ridirezionata dal Digital Adapter al Physical Adapter tramite il metodo `this.publishPhysicalAssetActionWldtEvent`.

4.3.3 Digital Adapter

L'ultimo componente che andremo a realizzare è il Digital Adapter, che svolge i seguenti compiti:

- Ricevere eventi dal DT Core relativi a variazioni di proprietà, eventi, azioni disponibili o relazioni.

- Esporre le informazioni ricevute al mondo esterno secondo il protocollo utilizzato per la sua implementazione.

- Gestire le azioni digitali e inoltrarle al Core affinché possano essere processate e validate dalla Shadowing Function.

Nell'implementazione dei Digital Twin realizzata per il progetto, sono stati utilizzati diversi tipi di Digital Adapter:

- Un MQTT Digital Adapter (fornito dalla libreria) per il Water Level Subsystem tramite il quale l'evento `alarm` viene pubblicato su un topic MQTT per renderlo disponibile al mondo esterno, in particolare allo Smart Light Subsystem, che è in ascolto sul Physical Adapter.

- Un MQTT Digital Adapter per lo Smart Light Subsystem, tramite il quale il DT aggiorna il corrispondente PA delle variazioni della proprietà `status`. In questo caso, il DT estende le funzionalità del PA, poiché non solo il PA determina lo stato del DT, ma anche viceversa.

- Per entrambi i sistemi, è stato creato un nuovo HTTP Digital Adapter che consente di esporre al mondo esterno le variazione delle proprie proprietà tramite Server-Sent-Events. Per il Water Level Subsystem, è stato esposto anche un endpoint tramite cui il Digital Adapter può ricevere richieste di azioni. Ai fini del progetto, non sono stati gestiti eventi e relazioni con questo adapter, poiché i due sistemi non ne necessitavano.

Implementazione HTTP Digital Adapter

La classe base che andremo ad estendere è `DigitalAdapter`, che accetta come tipo generico una Configuration che ne definisce il comportamento.

Lo scopo di una Configuration è rendere il Digital Adapter il più generico possibile in modo che possa essere riutilizzato anche con altri parametri, consentendo all'utente finale di impostare solo la configurazione desiderata. A tale scopo, sono state definite le due classi `HttpDigitalAdapterConfiguration` e `HttpDigitalAdapterConfigurationBuilder`, che permettendo di creare una nuova configurazione utilizzando il Builder Pattern. In particolare, la configurazione creata consente di definire un endpoint su cui ricevere un'azione specificata o di definire una lista di proprietà che, in caso di variazione, genereranno degli SSE per notificare il mondo esterno. Per avere accesso alla configurazione all'interno dell'adapter, si può utilizzare il metodo `getConfiguration()`.

La classe `DigitalAdapter` è dotata di molte callback che vengono notificate quando si verifica qualsiasi tipo di variazione nello stato del Digital Twin. Nell'implementazione realizzata, sono stati implementati i seguenti metodi:

- `onDigitalTwinSync(IDigitalTwinState digitalTwinState)`: questo metodo svolge un ruolo centrale, in quanto è quello in cui il Digital Adapter riceve la descrizione del Digital Twin State. Attraverso il parametro `digitalTwinState`, l'adapter può analizzare proprietà, azioni, eventi e relazioni, decidendo quali entità osservare. L'adapter, di default, osserva automaticamente le variazioni di stato, in termini di descrizione di proprietà, eventi, relazioni e azioni. Tuttavia, solo per le proprietà osserva automaticamente le variazioni del valore chiamando il metodo di callback `onStateChangePropertyUpdated`. Pertanto, per tutte le altre entità, è necessario avviare manualmente l'osservazione delle variazioni del valore utilizzando le funzioni corrispondenti all'interno di questo metodo.

- `onAdapterStart`: in questo metodo, viene creato il server web (realizzato tramite Undertow [16]) utilizzato per la comunicazione con il mondo esterno. Dopo aver specificato l'host e la porta su cui eseguire il server, vengono creati vari handler per la gestione delle richieste:

  - Per ogni azione specificata nella configurazione viene creato un endpoint con metodo POST, consentendo la ricezione di richieste di azioni e il loro inoltro alla Shadowing Function tramite il metodo `publishDigitalActionWldtEvent` in caso di successo. Per motivi di semplicità, non sono stati aggiunti controlli di correttezza dei campi inviati.
  - Per ogni insieme di liste di proprietà, viene creato un `ServerSentEventHandler` che consente ad agenti esterni di iscriversi all'endpoint desiderato per ricevere Server-Sent-Events in merito alle variazioni delle proprietà specificate.

- `onStateChangePropertyUpdated`: questo metodo viene chiamato quando si verifica una variazione di valore in una delle proprietà del DT State. Nell'implementazione, il metodo verifica quali SSE Handler sono associati alla proprietà aggiornata e, per ognuno di essi, se esistono delle connessioni attive, invia un evento con il valore della proprietà aggiornato.

4.3.4 Processo del Digital Twin

Dopo aver implementato tutti i componenti, definendo lo schema e le modalità di comunicazione tra il DT e il PA, tra il DT e la Dashboard e tra i DT stessi, non resta che creare una classe con un metodo `main`, all'interno del quale istanzieremo un WLDT Engine con le componenti create e avvieremo il DT (vedi Listato 4.8, semplificato per motivi di chiarezza).

```
package com.thesis;

import it.wldt.core.engine.WldtEngine;

public class WaterLevelSubsystem {
    public static void main(String[] args) {
        try{

         //Istanzia WLDT Engine con relativa Shadowing Function
            WldtEngine digitalTwinEngine = new WldtEngine(new
                WaterLevelSubsystemShadowingFunction("wls-sf"),
                "water-level-subsystem");

         //Aggiunge tutti i Digital e Physical Adapter
            digitalTwinEngine.addPhysicalAdapter(getMqttEspPA());
            digitalTwinEngine.addDigitalAdapter(getMqttEventDA());
            digitalTwinEngine.addDigitalAdapter(getHttpDA());
```

```
        //Avvia il DT
           digitalTwinEngine.startLifeCycle();

        }catch (Exception e){
           e.printStackTrace();
        }
    }

    private static MqttPhysicalAdapter getMqttEspPA(){
      MqttPhysicalAdapterConfiguration conf = ...;
        return new MqttPhysicalAdapter("wls-mqtt-esp",conf);
    }

    private static MqttDigitalAdapter getMqttEventDA(){
        MqttDigitalAdapterConfiguration conf = ...;
        return new MqttDigitalAdapter("wls-mqtt-event", conf);
    }

    private static HttpDigitalAdapter getHttpDA(){
        HttpDigitalAdapterConfiguration conf = ...;
        return new HttpDigitalAdapter("wls-http", conf);
    }
}
```

Listato 4.8: Creazione del DT del Water Level Subsystem

Capitolo 5

Confronto Finale tra i due framework

Dopo aver presentato un'analisi dei due framework e aver implementato il caso di studio utilizzando entrambi, in questo capitolo esamineremo le differenze riscontrate tra le due tecnologie, approfondiremo: in che modo vengono strutturati i DT, il tipo di architettura adottata, i diversi approcci di sviluppo proposti e l'interazione con il PA.

5.1 Architetture a confronto

Entrambi i framework sono progettati per supportare l'esecuzione dei Digital Twin sia su ambienti cloud che in locale. Tuttavia, le architetture alla base di questi due framework sono notevolmente diverse, influenzate dalle funzionalità specifiche e dal ruolo che svolgono come framework per la gestione dei DT.

Ditto è un framework concepito per la creazione e la gestione centralizzata dei DT. In questo contesto, tutti i DT risiedono all'interno di un'unica architettura centrale e comunicano tra di loro tramite un protocollo interno noto come Ditto Protocol. Per creare ed eseguire un DT in questo framework, è necessario utilizzare l'intera infrastruttura di Ditto, compresi moduli che potrebbero risultare superflui in alcune situazioni. Ditto si distingue per la sua flessibilità nella gestione di grandi quantità di dati, nonché per l'indicizzazione e la ricerca di informazioni specifiche relative a una Ditto Thing.

Al contrario, WLDT è una framework progettato per la creazione e la gestione distribuita dei DT. In questo approccio, ogni DT è considerato un'entità software indipendente che può comunicare con altri DT tramite vari protocolli. Un'altra caratteristica chiave di WLDT è la sua modularità, che facilita una gestione più semplice e leggera. Un'istanza WLDT, che rappresenta un

singolo DT, è costituita da un core multi-thread in grado di gestire diversi componenti in modo indipendente. La modularità consente agli sviluppatori di selezionare i componenti da utilizzare in base al comportamento del DT e alla sua relazione con la controparte fisica.

5.2 Modellazione del Digital Twin a confronto

Le prospettive di Ditto e WLDT sulla modellazione del PA nel relativo DT sono notevolmente divergenti. In Ditto, la concezione di un DT rimane di tipo monolitico, in cui tutta la complessità di un PA è gestita come un'entità software unica. D'altra parte, in WLDT, un PA composto da sotto-entità può essere frammentato in più DT indipendenti, ognuno responsabile delle proprietà e delle funzionalità di una sotto-entità. Questi DT possono interagire tra di loro creando un sistema aggregato. WLDT consente anche di tracciare le relazioni tra le varie sotto-entità attraverso il meccanismo delle **relationship**, un concetto assente in Ditto. Ciò consente di creare sistemi distribuiti di DT in grado di scambiarsi informazioni e dati.

Nel caso di studio considerato, l'utilizzo del concetto di **relationship** poteva essere applicato ai due sottosistemi, creando un terzo DT "aggregato" con due relazioni di tipo *"has-a"* con i DT dei due sottosistemi, che a loro volta possono contenere una relazione di tipo *"part-of"*. Questo approccio consente di creare un sistema di DT che può essere interrogato a diversi livelli di granularità. Ad esempio, accendendo al DT composto, si può ottenere accesso solo alle variabili di stato dei due sottosistemi, mentre accedendo ai DT di livello inferiore, si può ottenere accesso a funzionalità e proprietà più specifiche del relativo PA.

Un DT creato con Ditto non dispone di un concetto di modello o stato, a parte ciò che è contenuto nel campo `definition` di una Thing, che viene importato da un agente esterno come la WoT Thing Description. In contrasto, WLDT, pur non supportando ufficialmente la WoT Thing Description, ne implementa già molte funzionalità, fornendo un modello contenente un Digital Twin State che espone proprietà, azioni, eventi e relazioni.

Come mostrato nella Tabella 5.1, le due tecnologie utilizzano termologie diverse per mappare concetti simili. Tuttavia, tralasciando il concetto di relationship, i due approcci si sono dimostrati sufficientemente adeguati per modellare il PA proposto nel caso di studio.

Concetto Ditto	Concetto WLDT
Thing	Digital Twin Model
Attributes	Properties (non osservate)
Message diretto verso Thing	Actions
Message inviato da Thing	Events
Features	Properties (osservate)
x	Relationship

Tabella 5.1: Confronto termini usati in Ditto e WLDT

Trovo il modello proposto da WLDT più efficace, poiché consente di identificare con maggiore chiarezza le funzionalità e proprietà fornite dal DT. Contrariamente, Ditto senza l'integrazione con WoT non fornisce un modello chiaro, tuttavia tramite l'utilizzo dell'integrazione consente di esporre un modello quasi identico a quello di WLDT.

Una Ditto Thing, senza l'integrazione con WoT, espone chiaramente solo le proprietà di un PA (Feature e Attribute), mentre gli eventi e le azioni, mappati da Ditto nel concetto di Message, risultano meno chiari, costringendo applicazioni esterne e PA a utilizzare messaggi in Ditto Protocol.

Altre differenze riscontrate:

- Mentre WLDT attualmente non dispone di un sistema integrato per accedere allo stato passato del DT, Ditto permette, grazie all'integrazione con un servizio MongoDB, di accedere allo stato di un DT in uno specifico istante o a uno specifico numero di revisione.

- Ditto, a differenza di WLDT, non dispone di un sistema di tipizzazione. Pertanto, nonostante tramite il WoT sia possibile specificare un modello che definisce il tipo delle proprietà, Ditto non effettuerà alcun controllo per verificare che il tipo specificato sia quello corretto. Tale controllo è affidato al sistema che invia l'informazione.

5.3 Interazione con il Physical Asset a confronto

I due framework differiscono anche nella tipologia di iterazione con il PA e nella gestione dei DT.

Ditto considera i DT come entità passive che vengono create e poi subordinate ai servizi che ne gestiscono proprietà, dati e comportamento. Una Ditto Thing non ha un ciclo di vita ben definito ma agisce semplicemente come una

struttura dati per informazioni provenienti dal PA. Non viene utilizzato il concetto di processo di shadowing, inteso come un processo attivo che gestisce la sincronizzazione tra entità fisica e digitale, ma il DT attende passivamente che i dati del PA siano inviati su una connessione o tramite messaggi Ditto.

Al contrario, in WLDT un DT, proprio come un PA, ha un ciclo di vita ben definito, partendo da una fase di inizializzazione, per passare alla fase di sincronizzazione con il PA e terminare con un'eventuale dismissione del sistema. Questo approccio garantisce una gestione più flessibile del DT che implementa tutte le funzionalità e caratteristiche del corrispondente PA, con una gestione dinamica del modello e del suo stato. Utilizzando una gestione a stati del DT una qualsiasi variazione nel PA può essere mappata in un corrispondente azione nel DT. Ad esempio, un evento inviato dal PA verrà mappato in un Physical Event gestito dalla Shadowing Function che, attraverso i diversi metodi di callback, potrà far corrispondere a questo evento un'altra serie di azioni. Questo modello dinamico consente potenzialmente al DT di evolversi insieme al PA.

Ditto, a differenza di WLDT, offre di default dei meccanismi di autenticazione e autorizzazione, che consentano solamente ai soggetti autorizzati di accedere a una risorsa. Questi meccanismi risultano fondamentali per garantire la proprietà di ownership, poiché il DT deve essere in grado di ricevere informazioni soltanto da soggetti autorizzati, come PA e applicazioni esterne che sono autorizzate a comunicare con il DT. WLDT, al contrario, non fornisce un supporto predefinito per l'autenticazione e l'autorizzazione ma lascia allo sviluppatore la possibilità di eseguire controlli e prendere decisioni all'interno di Physical e Digital Adapter al momento della ricezione o prima dell'invio delle informazioni.

Nessuno dei due framework definisce un metodo generale per lo scambio di informazioni tra due DT, che siano appartenenti allo stesso framework o a framework diversi. Tuttavia, tale tipo di comunicazione non risulta completamente necessario, in quanto è compito dello sviluppatore definire uno schema di comunicazione in base alle proprie esigenze applicative. Un DT può essere trattato come qualsiasi altra applicazione IoT o PA, interagendo con altri DT attraverso i meccanismi già implementati: utilizzando messaggi nel caso di Ditto o creando una comunicazione tra Physical Adapter di un DT e Digital Adapter dell'altro, nel caso di WLDT.

5.4 Sviluppo del Digital Twin a confronto

Uno degli obiettivi del caso di studio era confrontare le modalità di sviluppo di un DT nei due framework, considerando la prospettiva di un programmatore

che si interfacciava con essi per la prima volta.

Entrambi gli strumenti hanno dimostrato un'efficacia equa nell'implementazione dei PA del caso di studio. In generale, ritengo che WLDT sia uno strumento più accessibile per un programmatore che si interfaccia ad esso per la prima volta, poiché presenta una struttura più semplice e definisce chiaramente i metodi necessari per ottenere il comportamento desiderato. D'altra parte, Ditto potrebbe risultare più complesso, poiché si basa su vari servizi esterni ed introduce protocolli proprietari che devono essere utilizzati dallo sviluppatore durante l'implementazione.

Dal punto di vista della programmazione, penso che il tipo di approccio sia diverso per i due framework: in Ditto, si "configura" il modello del DT attraverso diversi parametri per prepararlo a ricevere informazioni dal PA e dal mondo esterno, mentre in WLDT, proprio come per il PA, si programma un comportamento attivo del DT, consentendo la definizione di un modello più robusto ed efficace.

Alcune caratteristiche che hanno portato alle considerazioni precedenti sono:

- Ditto offre un servizio più completo, essendo un progetto di dimensioni maggiori, mette a disposizione vari SDK e API per la creazione e la gestione dei DT, insieme a diverse interfacce già implementate per interagire con il PA. Tuttavia, se uno sviluppatore volesse estendere il funzionamento di Ditto per scopi legati al suo progetto, sarebbe necessario utilizzare i servizi (Pekko, Docker, MongoDB) che Ditto stesso impiega per gestire i propri micro-servizi. Questa complessità potrebbe risultare ostica per chi si avvicina a Ditto per la prima volta. Inoltre, l'utilizzo del Ditto Protocol, seppur efficace come formato di comunicazione per la gestione dei DT e dei PA, presenta alcune limitazioni, a mio avviso:
  - Il protocollo non si limita alla comunicazione interna a Ditto, ma richiede che sia il PA che le applicazioni esterne lo utilizzino per l'invio di messaggi attraverso specifiche API. Ciò implica che sia il PA che l'applicazione esterna debbano adattarsi alle funzionalità del DT e non viceversa. L'adozione di un formato di messaggi standard e non proprietario potrebbe favorire una maggiore integrazione con diversi tipi di PA.
  - Ditto introduce meccanismi di mappatura del payload nelle connessioni consentendo a PA o applicazioni esterne di inviare messaggi in potenzialmente qualsiasi formato. Tuttavia, questi messaggi devono essere convertiti in messaggi Ditto Protocol per essere elaborati da Ditto, introducendo un overhead non necessario nella comunicazione, che, come evidenziato in [11], potrebbe impattare sulle pre-

stazioni generali della gestione delle informazioni e funzionalità del DT.

- WLDT, essendo un progetto di dimensioni minori e ancora in una fase iniziale rispetto a Ditto, fornisce l'integrazione per un numero limitato di protocolli per l'interazione con il PA o le applicazioni esterne. Tuttavia, l'architettura modulare del framework lo rende naturalmente estendibile: gli sviluppatori possono creare dei moduli personalizzati (Physical o Digital Adapter) per implementare il protocollo di comunicazione necessario per interagire con il proprio PA. I metodi di callback forniti da WLDT per rispondere a ogni cambiamento nello stato del DT rendono il framework facilmente configurabile, anche per sviluppatori inesperti o alle prime esperienze con la libreria. Non è richiesto un protocollo di comunicazione interno, poiché i DT sono concettualmente separati e comunicano tra loro tramite i rispettivi Digital e Physical Adapter utilizzando protocolli standard.

Conclusioni

Il percorso di sviluppo del progetto ha consentito di realizzare un sistema basato su Digital Twin, comprendendo tutte le fasi, della creazione del Physical Asset, all'individuazione delle proprietà e delle funzionalità da includere nel modello del Digital Twin, fino alla sua effettiva implementazione. L'analisi dell'interazione tra due DT ha evidenziato schemi diversi per la comunicazione e le modalità utilizzate per crearla dai due framework coinvolti.

Un aspetto cruciale che è emerso è l'importanza di definire un modello standard per garantire l'interoperabilità tra i sistemi fisici e virtuali. Molte soluzioni attuali forniscono modelli "proprietari" che possono limitare l'interazione tra sistemi. In questo contesto, ritengo che il WoT Thing Description [14] offra un valido supporto alla modellazione dei Digital Twin. Non solo espone le API di un sistema alle applicazioni esterne interessate all'interazione, ma fornisce anche un contesto semantico del Digital Twin alle applicazioni esterne, consentendo loro di comprendere il ruolo del DT nel dominio in cui è collocato. Il WoT Thing Description permette di definire proprietà, eventi, azioni e relazioni (appartenenza, estensione, composizione), contribuendo così a una modellazione completa e flessibile del Digital Twin.

Un altro aspetto critico emerso durante l'analisi del caso di studio è l'importanza di seguire un processo generale durante l'implementazione di un DT. Prima ancora di iniziare l'implementazione del PA, è fondamentale avere una chiara visione delle proprietà, degli eventi, delle azioni e delle relazioni che il sistema metterà a disposizione delle applicazioni esterne. Il DT dovrebbe essere modellato in modo da riflettere appieno le caratteristiche del PA, mantenendo il PA il più possibile agnostico rispetto al DT. Questo approccio consente al PA di interagire con diversi tipi di DT o altri servizi semplicemente esponendo i propri dati, senza adattandosi a implementazioni specifiche, rendendo il processo scalabile per un numero elevato di DT.

L'obbiettivo della tesi era condurre un confronto tra due framework, Eclipse Ditto e White Label Digital Twin (WLDT), non tanto per determinare il migliore tra i due, bensì per identificare gli approcci differenti utilizzati nella modellazione e gestione del Digital Twin. Questo confronto mirava anche a fornire spunti utili per la progettazione futura della piattaforma accademica

WLDT. Dallo studio è emerso che, pur con approcci distinti, entrambi i sistemi sono in grado di modellare efficacemente un PA in un DT.

Ditto si distingue per la gestione efficiente di grandi quantità di dati relativi al DT, consentendo l'accesso solo alle parti autorizzate e fungendo da framework che fornisce funzionalità di "device-as-a-service" per il PA. Tuttavia, questo si verifica a discapito di una gestione attiva del comportamento del DT, che risulta più come una struttura dati passiva per modellare il PA.

Al contrario, WLDT modella i DT come entità con comportamento attivo, caratterizzate da un ciclo di vita e da un modello più complesso, rendendo il DT flessibile, modulabile ed estendibile. Il DT implementato con WLDT consente di adottare un approccio "one-to-one", in cui il PA si interfaccia con un DT indipendente, alleggerendo il carico della comunicazione. WLDT permette di definire in modo più chiaro la relazione esistente tra DT e PA e di monitorarla costantemente. Inoltre, fornisce meccanismi nativi per definire relazioni di composizione, aggregazione ed appartenenza, semplificando la gestione di un sistema distribuito di DT.

Il confronto non ha indicato la necessità di sostituire uno dei due framework con l'altro, poiché questi ricoprono ruoli diversi nella gestione dei DT. Si potrebbe avanzare un'ipotesi di integrazione dei due sistemi: WLDT potrebbe occuparsi della gestione attiva del DT in relazione al PA, garantendo l'interoperabilità, mentre Ditto potrebbe essere integrato per archiviare i dati del DT in modo complesso, consentendo l'accesso anche ai suoi dati storici.

In conclusione, i Digital Twin si rivelano un paradigma vantaggioso per i sistemi fisici sia in ambito industriale sia nell'espansione del concetto di IoT, inteso come una serie di oggetti fisici che forniscono i propri servizi al mondo digitale. Questi consentono di ampliare le funzionalità di un sistema fisico, non solo esponendo flessibilmente i dati a terze applicazioni, ma anche creando una replica digitale per la verifica di guasti e la risoluzione di malfunzionamenti.

Ringraziamenti

Un ringraziamento al Prof. Alessandro Ricci, al Prof. Marco Picone e al Dott. Samuele Burattini per la loro disponibilità, guida e supporto durante la realizzazione di questa tesi.

Un ringraziamento speciale va ai miei genitori e alla mia famiglia per il costante sostengo lungo tutto il mio percorso accademico e per la fiducia che hanno sempre riposto in me.

Infine, desidero ringraziare Margherita per essere stata sempre al mio fianco e per il sostegno nei momenti più difficili del mio percorso.

Bibliografia

[1] Dashboard Web. https://github.com/Fab-Ver/SmartBridgeDashboard.

[2] Eclipse Ditto. https://eclipse.dev/ditto/index.html.

[3] FreeRTOS. https://www.freertos.org/index.html.

[4] Michael Grieves. *Virtually Perfect: Driving Innovative and Lean Products through Product Lifecycle Management*. Space Coast Press, 11 2011.

[5] Michael Grieves. Origins of the digital twin concept. Technical report, Florida Institute of Technology, 2016.

[6] JSON-LD. https://json-ld.org/.

[7] Roberto Minerva, Gyu Myoung Lee, and Noël Crespi. Digital twin in the iot context: A survey on technical features, scenarios, and architectural models. *Proceedings of the IEEE*, 108(10):1785–1824, 2020.

[8] MQTT Physical Adapter. https://github.com/wldt/mqtt-physical-adapter-java.

[9] Pekko Cluster. https://pekko.apache.org/docs/pekko/current/typed/cluster-concepts.html.

[10] Marco Picone, Marco Mamei, and Franco Zambonelli. WLDT: A general purpose library to build iot digital twins. *SoftwareX*, 13:100661, 2021.

[11] Marco Picone, Marco Mamei, and Franco Zambonelli. A flexible and modular architecture for edge digital twin: Implementation and evaluation. *ACM Trans. Internet Things*, 4(1), feb 2023.

[12] React. https://react.dev/.

[13] Smart Bridge. https://github.com/Fab-Ver/SmartBridgePA.

[14] Thing Description. https://www.w3.org/TR/wot-thing-description11/#introduction-td.

[15] Thing Model. https://www.w3.org/TR/wot-thing-description11/#introduction-tm.

[16] Undertow. https://undertow.io/.

[17] Web of Things. https://www.w3.org/groups/wg/wot.