ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

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

PROGETTAZIONE E SVILUPPO DELL'INTEGRAZIONE DI DIGITAL TWINS NEL WEB OF THINGS

Elaborato in
INTERNET OF THINGS

Relatore
Prof. ALESSANDRO RICCI

Corelatore
Dott. SAMUELE BURATTINI

Presentata da
PIETRO PASINI

Anno Accademico 2024 – 2025

Indice

Introduzione

L'evoluzione dei sistemi cyber-fisici e dell'Internet of Things ha prodotto un ecosistema caratterizzato da una frammentazione significativa in termini di protocolli di comunicazione, formati di rappresentazione dei dati e modelli architetturali. Questa eterogeneità, se da un lato riflette la diversità dei domini applicativi e dei requisiti specifici, dall'altro costituisce un ostacolo concreto alla realizzazione di sistemi interoperabili su larga scala, limitando le possibilità di integrazione e orchestrazione di dispositivi e servizi provenienti da ecosistemi differenti.

Due paradigmi emergenti affrontano aspetti complementari di questa problematica. Il Web of Things, standardizzato dal World Wide Web Consortium, propone un'astrazione unificata per l'interazione con dispositivi eterogenei attraverso protocolli web consolidati, definendo un vocabolario comune per descrivere le capacità dei dispositivi indipendentemente dalle tecnologie sottostanti. Parallelamente, nel dominio dei Digital Twin, che si è affermato come approccio fondamentale per la rappresentazione digitale di entità fisiche, sono state progettate e implementate architetture modulari che incapsulano in specifici componenti il problema dell'interoperabilità.

L'osservazione di queste proposte suggerisce che l'integrazione dei Digital Twin nel paradigma Web of Things rappresenterebbe un'opportunità per superare i requisiti di interoperabilità richiesti. Il presente lavoro di tesi esplora questa possibilità e la realizza attraverso lo sviluppo di componenti per un framework per la creazione di Digital Twin che realizzano un'architettura modulare.

La tesi si articola in quattro capitoli principali. Il primo capitolo introduce lo standard W3C Web of Things, analizzandone l'architettura, i modelli di interazione e l'idea centrale di Thing Description. Il secondo ripercorre l'evoluzione storica del concetto di Digital Twin, e ne descrive le tipiche applicazioni. Viene quindi esaminata un'architettura modulare per Digital Twin, caratterizzata dalla separazione tra interfacciamento fisico, logica di sincronizzazione ed esposizione digitale, che trova realizzazione concreta nel framework WLDT. Il terzo capitolo costituisce il nucleo concettuale della tesi, definendo i requisiti funzionali per l'integrazione tra Digital Twin e Web of Things. Vengono ana-

lizzate analogie e differenze nelle rappresentazioni di dispositivi IoT pensate dai due modelli, e si discutono strategie di mapping tra esse. Il quarto capitolo presenta l'implementazione dei componenti sviluppati per WLDT, descrivendo le scelte tecnologiche, l'architettura e i meccanismi implementati per aderire ai requisiti, in fine i test effettuati e i risultati ottenuti.

Capitolo 1

Web of Things W3C

1.1 L'idea del Web of Things

1.1.1 La storia

Il concetto di *Web of Things* (WoT) nasce dall'esigenza di estendere i principi e le tecnologie del Web al dominio dell'Internet of Things (IoT). La sua evoluzione si sviluppa lungo una traiettoria che parte da esigenze concrete di interoperabilità e accessibilità, per arrivare oggi a un insieme di standard consolidati e scenari applicativi in continua espansione.

Le necessità alla base del Web of Things

Con la diffusione massiva di dispositivi connessi agli inizi degli anni 2000, emerse chiaramente il problema della frammentazione tecnologica: produttori e programmatori adottavano i diversi protocolli di comunicazione (come ZigBee, Z-Wave o Modbus) senza un'infrastruttura a più alto livello, limitando la possibilità di integrazione trasversale. La crescente eterogeneità dei dispositivi rendeva sempre più complesso lo sviluppo di soluzioni scalabili e riutilizzabili. Per superare tali limiti, divenne cruciale iniziare a progettare un livello di astrazione comune, capace di esporre le funzionalità dei dispositivi attraverso interfacce pubblicamente comprensibili. La scelta cadde naturalmente sugli standard consolidati del Web, in grado di offrire un linguaggio universale e indipendente dalle specificità hardware.

Evoluzione

- **2000–2001 – le prime sperimentazioni**: ricercatori e sviluppatori iniziarono a proporre interfacce web elementari per il controllo di oggetti

fisici, utilizzando protocolli HTTP e pagine HTML. Un esempio emblematico è il progetto *Cooltown* di HP Labs (2000-2001) [1], che mirava a collegare oggetti e luoghi fisici con URL univoci: fu virtualizzato un museo, nel quale le opere avevano degli URL con le informazioni, ed una sala conferenze dove i servizi della stanza, quali stampanti e proiettori, erano esposti via web.

- **2007 – la formalizzazione del concetto**: il termine *Web of Things* venne formalmente definito da Dominique Guinard e Vlad Trifa, che fondarono la comunità webofthings.org e proposero un approccio RESTful per l'IoT [8]. Questo segnò l'inizio di un movimento verso l'utilizzo degli standard web come livello applicativo per l'Internet of Things.

- **2011–2014 – la standardizzazione embrionale**: iniziarono a consolidarsi approcci basati su architetture REST, JSON e tecnologie semantiche. Furono pubblicate le tesi di dottorato di Guinard [6] e Trifa [13], che delinearono modelli architetturali WoT e si diffusero le prime piattaforme middleware orientate al Web.

- **2014–2015 – l'ingresso del W3C**: il World Wide Web Consortium (W3C) organizzò un workshop sul Web of Things a Berlino nel giugno 2014, seguito dal lancio del *Web of Things Interest Group* nel gennaio 2015 [14]. Nel 2015 venne anche pubblicato il *Web Thing Model* come W3C Member Submission [7].

- **2016–2019 – sviluppo delle specifiche**: nel 2016-2017 il W3C avviò il *Web of Things Working Group* con l'obiettivo di sviluppare raccomandazioni formali. Nel 2019 furono pubblicate le versioni Candidate Recommendation della *WoT Architecture* e della *WoT Thing Description*.

- **2020–presente – maturità e adozione**: nell'aprile 2020, le specifiche *WoT Architecture* e *WoT Thing Description* diventarono raccomandazioni ufficiali del W3C [15, 16], segnando il riconoscimento del paradigma come standard globale. Nel dicembre 2023 è stata pubblicata la versione 1.1 della Thing Description. Le attività recenti si concentrano su sicurezza, discovery automatico, e integrazione con tecnologie emergenti.

1.1.2 Applicazioni del Web of Things

Oggi il *Web of Things* (WoT) trova applicazione in numerosi domini diversi, riflettendo la sua natura trasversale e di interoperabilità. Le architetture e gli standard proposti dal W3C permettono di integrare dispositivi e servizi

in scenari che spaziano dal contesto domestico a quello industriale, fino ad ambiti urbani e ambientali. Di seguito vengono presentati alcuni tra i casi più significativi.

Smart Home

Nel contesto *consumer*, la *smart home* rappresenta una delle applicazioni più diffuse del WoT. Attraverso l'uso di gateway e protocolli di comunicazione locale, i dispositivi domestici — quali sensori, telecamere, elettrodomestici e sistemi di climatizzazione — possono essere integrati in un ecosistema unificato. I dati raccolti vengono analizzati a livello di cloud, abilitando funzionalità avanzate come il controllo remoto, l'automazione basata su presenza o condizioni ambientali, la gestione dei consumi energetici e la manutenzione predittiva degli elettrodomestici. L'approccio WoT consente dunque di superare la frammentazione dei protocolli e di offrire agli utenti esperienze omogenee, migliorando comfort, efficienza energetica e sicurezza.

Smart Factory

In ambito industriale, il WoT costituisce un abilitatore chiave per il paradigma dell'Industria 4.0. Le *smart factory* integrano dispositivi e macchinari attraverso protocolli come Modbus, PROFINET, EtherCAT e OPC UA, che possono essere resi interoperabili tramite l'architettura WoT. L'uso di dispositivi edge consente di raccogliere e aggregare i dati provenienti da controllori di linea e macchine di produzione, rendendoli disponibili per servizi cloud di analisi, monitoraggio remoto e manutenzione predittiva. Ciò permette non solo di ridurre i costi legati ai fermi macchina, ma anche di migliorare la sicurezza dei lavoratori grazie al monitoraggio ambientale (ad esempio presenza di gas tossici o calore eccessivo) e di ottimizzare l'intera catena di produzione.

Smart Cities e altre applicazioni

Oltre ai contesti domestici e industriali, il WoT trova applicazione in numerosi altri ambiti emergenti. Nelle *smart cities*, esso abilita sistemi di monitoraggio per infrastrutture critiche, illuminazione pubblica, parcheggi intelligenti e gestione dei rifiuti, contribuendo a una gestione più sostenibile ed efficiente delle risorse urbane. Nel settore dei trasporti, i veicoli connessi possono sfruttare il WoT per condividere dati con servizi cloud e migliorare la sicurezza stradale. Altri domini rilevanti includono l'agricoltura di precisione, la sanità (ad esempio con il monitoraggio remoto dei pazienti) e il monitoraggio ambientale, in cui reti di sensori distribuiti inviano dati a piattaforme web per prevenire rischi ambientali o sanitari.

1.2 Modello

Secondo la definizione del W3C:

> *"Una Thing è un'astrazione di un'entità fisica o virtuale (ad esempio, un dispositivo o una stanza) descritta da metadati standardizzati".*

Normalmente questa entità è capace di esporre funzionalità e stati osservabili attraverso la rete tramite uno o diversi protocolli di telecomunicazioni. L'insieme dei metadati è precisamente esposto da un documento chiamato Thing Description, che rispecchia le specifiche del Web of Things. Tale descrizione può essere letta e interpretata da un sistema client che prende il nome di Consumer.

1.2.1 Rappresentazioni della Thing

Sono quindi emersi diversi punti di vista dell'oggetto Thing, che si distinguono con un lessico specifico:

- **Thing Description (TD)**: è un documento statico che specifica in modo formale le funzionalità, le interfacce e i metadati della *Thing*. La TD non esegue alcun comportamento, ma è l'elemento alla base del WoT, perché consente di comprendere come interagire con l'oggetto in modo standardizzato. Essa funge da contratto semantico e sintattico tra il dispositivo e i sistemi che vi accedono.

- **Exposed Thing**: rappresenta la controparte digitale effettivamente eseguita dall'oggetto o da una sua componente software. L'Exposed Thing è l'entità runtime che espone concretamente le proprietà, le azioni e gli eventi definiti nella TD, traducendoli in interfacce e protocolli di comunicazione.

- **Consumed Thing**: è la proiezione della *Thing* dal punto di vista di un Consumer. A partire dalla TD, esso genera un modello locale — la Consumed Thing — che fornisce metodi e strutture dati per interagire con l'Exposed Thing. Essa costituisce quindi l'adattamento della Thing Description ad un'istanza operativa utilizzabile dal software che la consuma.

1.2.2 Affordance e interazioni

Il concetto di *affordance*, introdotto inizialmente nella psicologia ecologica e successivamente ripreso da Donald Norman nell'ambito della Human-Computer Interaction, indica le proprietà percepite e reali di un oggetto che ne suggeriscono i possibili usi. Un esempio classico è la maniglia di una porta: la sua sola presenza suggerisce che la porta può essere aperta e, a seconda della forma, come debba essere utilizzata (ruotare, spingere verso il basso, ecc.).

Nel contesto del Web of Things, questo concetto assume un ruolo centrale nella *Thing Description*, poichè questa non si limita a descrivere le caratteristiche di una *Thing*, ma fornisce anche metadati leggibili da macchine che indicano chiaramente quali capacità l'oggetto offre e come esse possano essere sfruttate da un *Consumer*. Il W3C definisce quindi le *Interaction Affordances* come i meccanismi standardizzati che descrivono le possibili modalità di interazione con una *Thing*. Esse si distinguono in tre categorie fondamentali:

- **Proprietà (Properties)**: descrivono lo stato interno della *Thing* che, a seconda dei casi, può essere letto o modificato. Una proprietà corrisponde a un attributo persistente, ad esempio la temperatura corrente di un sensore o il valore soglia che attiva un allarme.

- **Azioni (Actions)**: rappresentano operazioni invocabili sulla *Thing* dall'esterno in maniera asincrona, che possono produrre effetti immediati o complessi. Le azioni catturano la dimensione procedurale dell'interazione, come accendere una lampadina, avviare un motore o aggiornare un firmware.

- **Eventi (Events)**: modellano notifiche asincrone generate dalla *Thing* in risposta a cambiamenti di stato o condizioni specifiche. Un esempio tipico è la segnalazione di un allarme, il rilevamento di un movimento o la notifica di avvio di una procedura automatica.

Questi tre pattern di interazione formano un vocabolario minimo in grado di modellare ogni forma di scambio tra un sistema e un oggetto connesso. Dal punto di vista del modello di comunicazione, le *Properties* sono spesso percepite come interazioni sincrone di lettura/scrittura di stati, anche se possono supportare notifiche di aggiornamento asincrone. Le *Actions* e gli *Events*, invece, rispecchiano più fedelmente il paradigma asincrono: le azioni come invocazioni dal sistema verso l'oggetto, gli eventi come notifiche dall'oggetto verso il sistema.

1.2.3 Il Servient

L'architettura del Web of Things definisce un insieme di componenti software e pattern progettuali che consentono l'implementazione pratica del paradigma WoT. Al centro di questa architettura si colloca il *Servient*, termine che deriva dalla contrazione di *Server* e *Client*.

Il modello di Servient

Un Servient è un'entità software che implementa l'interfaccia di runtime del W3C WoT e può operare contemporaneamente come produttore e consumatore di *Thing*. Questa duplice natura lo distingue dai tradizionali paradigmi client-server, rendendolo un componente altamente versatile e adattabile a diversi contesti applicativi. Dal punto di vista architetturale, un Servient è composto da due macro-componenti principali:

- **WoT Runtime**: è il nucleo esecutivo che fornisce le API standardizzate per la gestione delle *Thing*. Il runtime si occupa della gestione del ciclo di vita delle istanze (sia Exposed che Consumed), della serializzazione e deserializzazione delle Thing Description, e del coordinamento tra i diversi livelli architetturali.
- **Protocol Bindings**: costituiscono lo strato di adattamento tra il modello astratto delle affordance e i protocolli di rete concreti. Ogni binding implementa la logica necessaria per tradurre le operazioni WoT (readproperty, invokeaction, etc.) in chiamate specifiche del protocollo sottostante (HTTP, CoAP, MQTT, Modbus, etc.).

La natura modulare del Servient permette di aggiungere o rimuovere dinamicamente protocol bindings in base alle esigenze operative, garantendo estensibilità e adattabilità a contesti eterogenei.

Ruoli operativi e API standard

Un Servient può operare secondo due diverse modalità a seconda dello scenario applicativo, anche entrambe simultaneamente. Il W3C ha standardizzato un insieme di API che i Servient devono esporre per garantire interoperabilità a livello applicativo, distinguendo chiaramente tra operazioni di produzione e consumo.

Modalità Producer Quando opera come *producer*, il Servient ospita una o più Exposed Thing rendendole accessibili attraverso la rete. La WoT Scripting API fornisce il metodo `produce()`, che accetta una Thing Description e restituisce un'istanza di Exposed Thing. In questa modalità, il Servient:

- Istanzia e mantiene attive le Exposed Thing basate su Thing Description fornite;
- Permette di registrare handler per le affordance tramite metodi come `setPropertyReadHandler()`, `setActionHandler()` ed `emitEvent()`;
- Implementa la logica applicativa che collega le affordance esposte alle risorse fisiche o virtuali sottostanti;
- Gestisce le richieste in arrivo applicando le policy di sicurezza definite nella TD;
- Pubblica le Thing Description, aggiungendo negli appositi campi gli endpoint disponibili per accedere alle funzionalità.

Modalità Consumer In modalità *consumer*, il Servient utilizza Thing Description di oggetti remoti per interagire con essi. Il metodo `consume()` accetta una Thing Description e genera una Consumed Thing, ovvero un proxy locale che incapsula la comunicazione remota. Le responsabilità in questo ruolo includono:

- Acquisizione e parsing delle Thing Description da discovery service, repository o direttamente dai producer;
- Creazione di proxy che espongono metodi asincroni come `readProperty()`, `writeProperty()`, `invokeAction()` e `subscribeEvent()`, astraendo completamente i dettagli del protocollo di comunicazione;
- Gestione automatica delle credenziali e dei meccanismi di autenticazione richiesti;
- Implementazione della logica di retry, caching e gestione degli errori di rete;
- Orchestrazione di interazioni complesse che coinvolgono multiple *Thing*.

Queste API permettono agli sviluppatori di scrivere applicazioni WoT in modo completamente indipendente dal protocollo sottostante, delegando al runtime del Servient la gestione dei dettagli di comunicazione. Le implementazioni possono inoltre estendere le API standard con funzionalità di discovery (metodi come `discover()` o `exploreDirectory()`) per individuare automaticamente *Thing* disponibili nella rete locale o in directory remoti.

1.3 Struttura della Thing Description

La Thing Description (TD) costituisce il documento fondamentale attraverso cui una *Thing* espone le proprie capacità e metadati in forma standardizzata e machine-readable. Come definito dal W3C, essa rappresenta il contratto informativo che abilita l'interoperabilità nel Web of Things, permettendo ai *Consumer* di scoprire, comprendere e interagire con dispositivi eterogenei senza conoscerne a priori i dettagli implementativi.

1.3.1 Formato del documento

Una Thing Description è rappresentata mediante un documento JSON-LD (JSON for Linking Data), che combina la semplicità sintattica del formato JSON con le capacità semantiche del Linked Data, questo garantisce facilità di elaborazione automatica dai parser standard implementati nei diversi linguaggi. Successivamente viene esaminata la struttura dei campi principali secondo le specifiche del W3C, concentrandosi su quelli più utili a questo lavoro.

1.3.2 Metadati generali

I metadati di alto livello includono informazioni identificative e descrittive della *Thing*:

- **@context**: definisce i vocabolari semantici utilizzati nel documento, tipicamente includendo il contesto WoT standard e eventuali estensioni di dominio;
- **id**: un identificatore univoco (URI) della *Thing*;
- **title**: nome leggibile dell'oggetto;
- **description**: descrizione testuale delle funzionalità;
- **@type**: classificazione semantica della *Thing* (opzionale).

Questi campi consentono ai sistemi di catalogare e ricercare dispositivi in base a criteri semantici, facilitando scenari di discovery automatico. La sezione `securityDefinitions` specifica inoltre i meccanismi di autenticazione richiesti (basic, bearer, apikey, etc.), che possono poi essere referenziati nelle singole affordance.

1.3.3 Interaction Affordances: specifica dettagliata

Il nucleo operativo della TD è costituito dalle tre categorie di affordance: **properties**, **actions** ed **events**. Ciascuna affordance è identificata da una chiave univoca all'interno del proprio namespace e descritta da un insieme di metadati che ne specificano le caratteristiche operative.

Data Schema

Prima di descrivere le singole affordance, è fondamentale comprendere come vengono specificati i tipi di dato. Il WoT adotta uno schema basato su JSON Schema, che permette di descrivere non solo tipi primitivi (`string`, `number`, `integer`, `boolean`, `null`), ma anche strutture complesse.

Per i tipi **object**, lo schema richiede i campi:

- **type**: `"object"`
- **properties**: un dizionario che associa a ciascun campo il proprio schema
- **required**: array dei campi obbligatori (opzionale)

Per i tipi **array** include invece:

- **type**: `"array"`
- **items**: lo schema degli elementi contenuti nell'array
- **minItems/maxItems**: vincoli sulla cardinalità (opzionali)

Questa composizione permette di descrivere strutture dati arbitrariamente complesse e annidate, essenziali per modellare scenari reali in cui una *Thing* gestisce informazioni articolate.

Properties

Ogni proprietà espone lo stato interno della *Thing* e viene descritta dai seguenti campi principali:

- **type**: schema del tipo di dato, che può essere primitivo, oggetto o array secondo quanto descritto precedentemente;
- **readOnly**: se `true`, la proprietà è accessibile solo in lettura;
- **writeOnly**: se `true`, la proprietà è accessibile solo in scrittura;
- **observable**: indica se la proprietà supporta notifiche asincrone di cambiamento;

- **unit**: unità di misura (opzionale), ad esempio `"celsius"` o `"percent"`;
- **forms**: array di form che specificano come accedere alla proprietà.

Una proprietà può quindi rappresentare sia dati semplici (ad esempio una temperatura espressa come numero), sia strutture complesse. Ad esempio, una proprietà che descrive la configurazione di rete di un dispositivo potrebbe essere di tipo `object` con campi per indirizzo IP, subnet mask e gateway.

Actions

Le azioni rappresentano operazioni invocabili sulla *Thing* e sono descritte da:

- **input**: schema del payload di input (opzionale)
- **output**: schema della risposta prodotta dall'azione (opzionale);
- **safe**: se `true`, l'azione non modifica lo stato del sistema;
- **idempotent**: se `true`, invocazioni multiple producono lo stesso effetto di una singola invocazione;
- **synchronous**: indica se l'azione viene eseguita in modo sincrono o asincrono;
- **forms**: array di form per l'invocazione.

Events

Gli eventi modellano notifiche asincrone generate dalla *Thing* e prevedono:

- **data**: schema del messaggio di notifica, che può contenere strutture complesse per veicolare informazioni articolate;
- **subscription**: schema dei parametri per la sottoscrizione (opzionale);
- **cancellation**: schema per la cancellazione della sottoscrizione (opzionale);
- **forms**: array di form per gestire la sottoscrizione agli eventi.

1.3.4 Forms e Protocol Bindings

Gli oggetti **form** rappresentano il ponte tra il modello astratto della TD e le implementazioni concrete dei protocolli di rete. Ogni form contiene:

- **href**: URI dell'endpoint di accesso;
- **contentType**: tipo MIME del payload (es. `"application/json"`);
- **op**: operazione o array di operazioni supportate;
- parametri specifici del protocollo (es. `htv:methodName` per HTTP).

Le operazioni possibili variano in base al tipo di affordance:

- Properties: `readproperty`, `writeproperty`, `observeproperty`, `unobserveproperty`
- Actions: `invokeaction`, `queryaction`, `cancelaction`
- Events: `subscribeevent`, `unsubscribeevent`

Una singola affordance può esporre più form, consentendo l'accesso tramite protocolli diversi (HTTP, CoAP, MQTT) per adattarsi a contesti operativi differenti. Ad esempio, una proprietà potrebbe essere leggibile via HTTP GET e osservabile tramite MQTT, offrendo flessibilità al *Consumer* nella scelta del meccanismo più appropriato.

1.3.5 Validazione e conformità

Il W3C fornisce schemi JSON Schema per la validazione sintattica delle TD, garantendo che i documenti prodotti rispettino le specifiche ufficiali. La conformità può essere verificata attraverso tool dedicati che controllano la presenza dei campi obbligatori, la correttezza dei tipi di dato e la coerenza dei riferimenti interni. Questa fase di validazione è fondamentale per assicurare l'interoperabilità reale tra produttori e consumer di diversi fornitori.

Capitolo 2

Digital Twin e WLDT

2.1 Definizione e applicazioni

2.1.1 Origini dell'idea

Il concetto di *Digital Twin* (DT) nasce nei primi anni 2000 nell'ambito del *Product Lifecycle Management* (PLM). Nel 2002, Michael Grieves presentò all'Università del Michigan il cosiddetto *Mirrored Spaces Model*, che proponeva la coesistenza di due spazi: uno fisico e uno virtuale, connessi da flussi informativi bidirezionali. L'idea di fondo era che ogni sistema reale avesse un gemello digitale, capace di accompagnarlo lungo tutto il ciclo di vita: creazione, produzione, utilizzo e dismissione [5].

2.1.2 Definizione

La terminologia di *Digital Twin* si è consolidata a partire dal 2010, quando NASA iniziò ad adottarla nei propri documenti strategici. Nel 2012 Glaessgen e Stargel pubblicarono la prima definizione ufficiale in ambito aerospaziale:

> *"A Digital Twin is an integrated multiphysics, multiscale, probabilistic simulation of an as-built vehicle or system that uses the best available physical models, sensor updates, fleet history, etc., to mirror the life of its corresponding flying twin."* [3]

Tale definizione, sebbene fortemente legata a questo contesto, segnò un punto di svolta, spostando il concetto dal PLM verso la modellazione predittiva e la manutenzione basata sui dati.

Negli anni successivi, il termine si è evoluto, arricchendosi di nuove interpretazioni, la pluralità di definizioni ha generato confusione, talvolta portando

a identificare erroneamente semplici modelli digitali o simulazioni come Digital Twin. Una successiva chiarificazione è stata introdotta in un articolo ieee del 2020, che ha distinto tre categorie concettuali in base agli scambi di informazioni [2]:

1. *Digital Model*: rappresentazione digitale statica di un oggetto fisico o pianificato, senza scambio automatico di dati;

2. *Digital Shadow*: replica digitale con flusso dati unidirezionale dal fisico al virtuale;

3. *Digital Twin*: integrazione bidirezionale in tempo reale, in cui modifiche a uno dei due domini (fisico o virtuale) si riflettono immediatamente sull'altro.

Questa classificazione ha contribuito a riportare il termine Digital Twin in linea con la sua origine legata al PLM, distinguendolo chiaramente da modelli virtuali e simulazioni tradizionali.

2.1.3 Applicazioni del Digital Twin

Grazie all'avanzamento dell'*Internet of Things* (IoT), dell'*Industrial IoT* e dell'intelligenza artificiale, le applicazioni del Digital Twin si stanno diffondendo in diversi settori [2].

Smart Cities

Nel contesto delle *smart cities*, i Digital Twins vengono utilizzati per modellare infrastrutture urbane, servizi energetici e reti di trasporto. L'integrazione di sensori IoT consente di raccogliere dati in tempo reale sul traffico, sui consumi energetici e sulla gestione dei rifiuti. Ciò permette di creare piattaforme digitali capaci di simulare scenari alternativi e ottimizzare decisioni strategiche, con benefici significativi in termini di sostenibilità, sicurezza e qualità della vita dei cittadini.

Industria 4.0

Il settore manifatturiero rappresenta uno dei principali ambiti di adozione dei Digital Twins, strettamente collegato al paradigma dell'Industria 4.0. In questo contesto, i gemelli digitali consentono di monitorare in tempo reale le prestazioni delle macchine e delle linee produttive, di effettuare manutenzione predittiva e di ridurre i tempi e i costi di produzione. Inoltre, permettono

di testare virtualmente modifiche ai processi produttivi prima della loro implementazione fisica, riducendo il rischio di errori e aumentando la flessibilità industriale.

Sanità e Healthcare

Nel settore sanitario, i Digital Twins offrono prospettive innovative per la medicina personalizzata e la gestione delle strutture ospedaliere. Esempi di applicazione includono la simulazione dell'effetto dei farmaci su pazienti virtuali, la pianificazione di interventi chirurgici e la gestione predittiva delle apparecchiature mediche. In prospettiva, la creazione di gemelli digitali di organi o dell'intero corpo umano potrebbe fornire strumenti potenti per la diagnosi precoce e il supporto alle decisioni cliniche.

2.1.4 Sintesi

Il Digital Twin si configura dunque come una tecnologia abilitante di fondamentale importanza per settori chiave quali smart cities, manifattura e sanità. La sua peculiarità consiste nell'integrazione continua tra il mondo fisico e quello virtuale, resa possibile dall'IoT e dall'analisi avanzata dei dati, con l'obiettivo di migliorare predittività, efficienza e capacità decisionale.

2.2 Architettura modulare per Digital Twin

2.2.1 Modello multidimensionale

Nel 2015 Grieves espone l'idea astratta di architettura scomposta in tre livelli separati. Secondo l'articolo [4], il modello concettuale del Digital Twin comprende tre parti principali:

1. prodotti fisici nello spazio reale
2. prodotti virtuali nello spazio virtuale
3. le connessioni di dati e informazioni che legheranno insieme i prodotti virtuali e quelli reali.

Questa ispirò una delle architetture più riconosciute per la modellizzazione di Digital Twin, il *modello a cinque dimensioni* (5D) proposto da Tao et al. [12]. Il modello 5D rappresenta formalmente un DT come una quintupla:

$$DT = (PE, VE, Ss, DD, CN)$$

dove:

- PE (*Physical Entity*): rappresenta l'entità fisica, costituita dai vari sottosistemi funzionali e dispositivi sensoriali che eseguono compiti predefiniti e raccolgono dati sullo stato operativo;
- VE (*Virtual Entity*): costituisce un modello digitale ad alta fedeltà dell'entità fisica, che integra geometrie, proprietà fisiche, comportamenti e regole.
- Ss (*Services*): include i servizi per PE e VE, ottimizzando le operazioni dell'entità fisica e garantendo l'alta fedeltà del modello virtuale attraverso la calibrazione continua dei parametri. Ogni servizio è caratterizzato da funzione, input, output, qualità e stato;
- DD (*DT Data*): mantiene tutti i dati utili al modello, provenienti dalla PE, generati dalla VE, usati dai servizi e relativi al dominio.
- CN (*Connection*): gestisce le connessioni bidirezionali tra le varie dimensioni (PE-VE, PE-DD, VE-DD, PE-Ss, VE-Ss, Ss-DD).

Questo modello si distingue dall'architettura originale di Grieves per la fusione esplicita di dati provenienti da aspetti sia fisici che virtuali, consentendo una cattura di informazioni più completa e accurata. Inoltre, incapsula le funzioni del DT (rilevamento, giudizio, predizione) come servizi per una gestione unificata e un utilizzo on-demand.

Nel recente articolo [10] è spiegato come le 5 dimensioni del modello di Tao possano essere mappate in una struttura software con tre componenti:

- **Physical Interface (PI)**: si occupa della connessione e sincronizzazione con l'entità fisica (PE), gestendo l'acquisizione dei dati dal mondo reale;
- **Model (M)**: elabora i dati ricevuti dalla PI implementando l'entità virtuale (VE), assolvendo alla digitalizzazione del gemello fisico e memorizzando i dati rilevanti (DD);
- **Digital Interface (DI)**: espone i risultati dei modelli del DT, insieme agli endpoint per richieste di azione e query verso entità esterne come servizi (SS), creando una connessione tra il DT e le applicazioni digitali.

Questa compattazione mantiene le funzionalità fondamentali del modello 5D organizzandole in una struttura software più snella. Nelle sezioni successive, ciascuno di questi tre componenti viene esaminato nel dettaglio, con particolare attenzione all'approccio modulare basato su adapter e l'interoperabilità che essi consentono.

2.2.2 Model

Il Model rappresenta il nucleo concettuale del Digital Twin, contenendo la rappresentazione digitale ad alta fedeltà dell'entità fisica. Come descritto nell'architettura proposta da Ricci et al. [11], il Model si basa su un modello M della corrispondente entità fisica, che definisce come quest'ultima viene rappresentata a livello digitale.

Stato del Digital Twin

Lo stato del DT costituisce la componente dinamica del modello e viene calcolato come funzione dello stato del Physical Twin (PT). Dato un modello M, lo stato dinamico S_{DT} di un DT può essere definito formalmente come una quadrupla:

$$S_{DT} = \langle P, R, E, A \rangle$$

dove:

- P rappresenta l'insieme corrente delle **Properties**: dati etichettati che cambiano dinamicamente con l'evoluzione dello stato della controparte fisica. Le proprietà rappresentano gli attributi osservabili del PT come valori di dati (variabili) che possono cambiare secondo l'evoluzione dello stato del PT;

- R rappresenta l'insieme corrente delle **Relationships**: connessioni del PT con altri physical asset, rappresentate come collegamenti ad altri digital twin. Come le proprietà, anche le relazioni sono osservabili, possono essere create dinamicamente e cambiare nel tempo. Diversamente dalle proprietà, non riguardano puramente lo stato locale del PT, ma permettono di riferirsi ad altri PT attraverso i corrispondenti DT;

- E rappresenta la sequenza degli **Events** generati fino al momento corrente: segnali non persistenti che catturano eventi rilevanti e osservabili occorsi al PT, a livello di dominio. Gli eventi sono generati dalle informazioni raccolte dall'asset fisico associato;

- A: rappresenta l'insieme delle **Actions**, ossia le possibili operazioni che il DT consente di invocare per conto della controparte fisica, sia per inviare feedback all'entità fisica che per sfruttare un servizio esposto dal DT.

Processo di Shadowing

Lo *shadowing* (o *twinning*) è il processo fondamentale che mantiene lo stato del DT S_{DT} sincronizzato con lo stato del PT S_{PT}, secondo il modello M. Questo processo opera in modo bidirezionale, garantendo sia la sincronizzazione dallo spazio fisico a quello digitale, sia l'attuazione di azioni dallo spazio digitale a quello fisico attraverso 2 funzioni di shadowing (2.1), (2.2). [11]

Shadowing da PT a DT

1. Qualsiasi cambiamento rilevante dello stato S_{PT} che avviene nel PT viene catturato da un evento ev_{PT};

2. L'evento ev_{PT} viene propagato al DT attraverso la Physical Interface;

3. Dato l'evento ev_{PT},il nuovo stato S'_{DT} del DT viene calcolato viene aggiornato mediante una *shadowing function* $Shad_{PT \to DT}$ che dipende dal modello M:

$$S'_{DT} = Shad_{PT \to DT}(S_{DT}, ev_{PT}) \tag{2.1}$$

   dove rappresenta il nuovo stato del DT dopo l'aggiornamento.

In sistemi concreti, i PT possono essere entità complesse con uno stato strutturato e distribuito. Il processo di shadowing può quindi coinvolgere multiple sorgenti che generano flussi di informazioni ed eventi. Le sorgenti possono includere anche altri DT, ovvero un DT può rappresentare un PT logico di alto livello (ad esempio, il DT di un'organizzazione) aggregando informazioni ed eventi forniti da altri DT.

Shadowing da DT a PT

1. Un'azione a_{DT} viene richiesta sul DT, ad esempio attraverso l'API del digital twin esposta dalla Digital Interface;

2. Una nuova richiesta di azione a_{DA} per il PT viene generata mediante un'ulteriore shadowing function $Shad_{DT \to PT}$:

$$a_{PA} = Shad_{DT \to PT}(S_{DT}, a_{DT}) \tag{2.2}$$

   e propagata al PT attraverso la Physical Interface;

3. La richiesta di azione a_{DT} viene applicata al PT, determinando un cambiamento dello stato del PT S_{PT}.

È importante sottolineare che una richiesta di azione a_{DT} non modifica direttamente S_{DT}. I cambiamenti a S_{DT} sono causati unicamente dallo shadowing da PT a DT, questo garantisce che il DT rimanga sempre una rappresentazione fedele dello stato effettivo del sistema fisico.

Gli aggiornamenti di stato vengono quindi comunicati alle applicazioni attraverso la Digital Interface, realizzando l'obiettivo del DT di fornire una rappresentazione aggiornata e sincronizzata del Physical Twin. Questo meccanismo di sincronizzazione bidirezionale permette al DT non solo di osservare passivamente lo stato del sistema fisico, ma anche di interagire attivamente con esso attraverso l'invocazione di azioni.

2.2.3 Physical Interface

Una delle principali sfide nella progettazione di Digital Twin risiede nella necessità di interfacciarsi con un mondo fisico eterogeneo, caratterizzato da dispositivi IoT che operano su standard, protocolli e architetture differenti.

Per affrontare questa eterogenità fisica, l'architettura propone un approccio modulare basato su **Physical Adapters (PA)**: moduli specializzati capaci di interagire con il PT utilizzando diversi protocolli e formati di dati. Ciascun PA è responsabile di mediare l'interazione bidirezionale attraverso un singolo canale di comunicazione (aggiornamento dello stato, acquisizione eventi e invocazione di azioni), semplificando la progettazione e la riutilizzabilità del componente e rendendolo configurabile per adattarsi a diversi contesti applicativi.

La Physical Interface diventa quindi un contenitore di diversi PA, con la responsabilità di gestirli e assicurare che il modello del DT possa interpretare, processare e sfruttare accuratamente i dati generati dal mondo fisico per creare la replica digitale e implementare i suoi comportamenti. Sebbene possa accadere che un PT invii tutti i dati relativi alla sua digitalizzazione attraverso un solo canale, è molto più frequente costruire un DT aggregando diverse sorgenti di informazione [10].

Physical Asset Descriptions

Per facilitare il processo di gestione dei diversi PA, viene introdotto il concetto di **Physical Asset Description (PAD)**: una descrizione delle capacità rese disponibili da un canale di comunicazione in termini di proprietà, azioni, eventi e relazioni che caratterizzano il PT associato. Ogni PA, poiché incapsula le caratteristiche di un canale, può generare la corrispondente PAD, disaccoppiando efficacemente la funzionalità dell'asset dai protocolli di comunicazione specifici utilizzati.

L'implementazione della generazione della PAD può risultare difficoltoso a causa delle diverse capacità dei protocolli di comunicazione, ma confinare questa complessità all'interno della PI consente al modello del DT di essere agnostico rispetto all'eterogeneità del mondo fisico sottostante.

La PAD permette quindi alla PI di scoprire, estrarre e gestire le informazioni dell'asset e presentarle al modello, che può scegliere quali sono rilevanti per l'implementazione del comportamento del DT. Per esempio, per creare il DT di un sensore di temperatura che trasmette dati binari su MQTT, la PI potrebbe essere composto da un adapter MQTT generico, configurato per analizzare correttamente il payload come numero decimale, e generare una PAD che pubblichi la proprietà temperatura disponibile al modello del DT come valore in gradi Celsius [10].

2.2.4 Digital Interface

Per favorire l'interoperabilità, i DT devono esporre o una Digital Interface (DI) standardizzata e general-purpose che possa servire diverse applicazioni, oppure - seguendo gli stessi principi di progettazione adottati per affrontare l'interoperabilità fisica - avere un'interfaccia modulare che possa soddisfare le diverse esigenze di applicazioni differenti.

Analogamente ai PA, l'architettura prevede che la DI sia composta da **Digital Adapters (DA)** modulari. Utilizzando il concetto di DA, la DI può esporre lo stato e i servizi del DT supportando molteplici formati di dati e pattern di interazione. Questo rende lo sviluppo dell'applicazione più semplice perché l'aggiunta di un DA specifico per l'applicazione non richiederebbe interventi nel resto del sistema. Essa risulterebbe più robusta e stabile poiché dipenderebbe solo dal DT, e i cambiamenti alla configurazione fisica del PT (ad es., aggiornamenti software, sostituzione di sensori, riconfigurazioni di rete) non avrebbero impatto sul software dell'applicazione. Anche se il PT dovesse cambiare (ad es., un aggiornamento software su un robot modifica il formato della telemetria), la DI del DT potrebbe rimanere invariata, poiché le modifiche avverrebbero entro i confini della PI e del modello del DT.

Tale modularità favorirebbe, inoltre, anche la connessione con applicazioni già esistenti e sistemi legacy. Solitamente interfacciarsi con questi ultimi implica avere poco controllo sui requisiti di integrazione, rendendo vantaggioso avere DT più flessibili per adattarsi meglio ai protocolli dell'applicazione.

Per esempio, utilizzando diversi DA, un DT potrebbe :

- esporre il suo stato corrente utilizzando diversi formati di rappresentazione,

- supportare sia meccanismi request-response che publish-subscribe per accedere ai suoi stati correnti e precedenti,

- supportare diversi linguaggi di query per accedere allo stesso data store.

Dunque, attraverso questo meccanismo, i DT raggiungono efficacemente il loro ruolo di ponte, proteggendo le applicazioni dalla complessità dei deployment fisici [10].

Digital Twin Descriptions

Un ulteriore livello di interoperabilità è possibile quando si consente ai DT di descrivere la propria DI, pubblicizzando capacità e canali di comunicazione disponibili che le applicazioni possono sfruttare. Un DT dovrebbe quindi utilizzare una **Digital Twin Description (DTD)**, che, similmente alla PAD, consentirebbe di rappresentare le caratteristiche del DT ai suoi osservatori.

Il modo in cui tali descrizioni sono implementate può differire significativamente, ma essendo rivolte a consumatori esterni, dovrebbero preferibilmente aderire a formati e rappresentazioni standard per essere utile nella pratica nel raggiungimento dell'interoperabilità [10].

2.2.5 Digital Twin lifecycle

Il DT è un'entità software "vivente" che attraversa diverse fasi dalla creazione alla disattivazione, questa dinamicità prende il nome di lifecycle e si può rappresentare con un diagramma a stati finiti (figura 2.1).

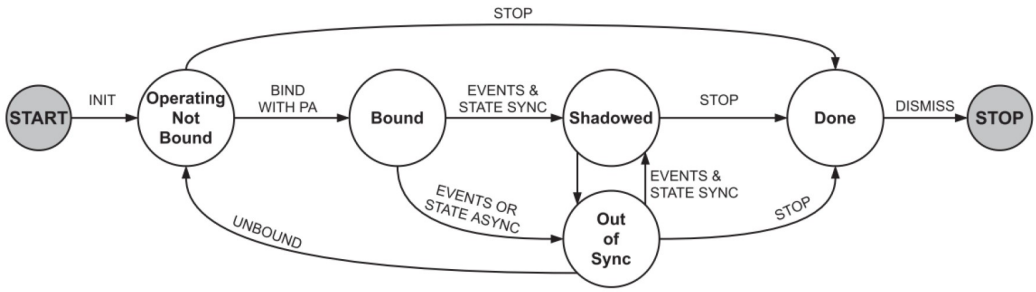

Figura 2.1: Lifecycle di un DT [11]

1. **Not Bound (Unbound)**: il DT si trova a seguito della fase di inizializzazione, indicando che tutti i moduli interni del DT sono attivi ma non c'è ancora un'associazione con il PA corrispondente;

2. **Bound**: il DT transita a seguito della corretta esecuzione della procedura di binding. La procedura di binding permette di connettere le due parti e abilita il flusso bidirezionale di eventi;

3. **Synchronized**: il processo di shadowing inizia e il suo stato è correttamente sincronizzato con quello del PA;

4. **Out of Sync**: determina la presenza di errori nel processo di shadowing. In questo stato, il DT non è in grado di gestire né eventi di allineamento dello stato né quelli generati dal livello applicativo;

5. **Done**: il DT raggiunge quando il processo di shadowing è arrestato, ma il DT continua ad essere attivo per gestire richieste provenienti da applicazioni esterne.

Il progetto modulare della PI ha un impatto sul ciclo di vita del DT, illustrato nella Figura 2.1. In particolare, deve affrontare le sfide delle transizioni dallo stato *Unbound* allo stato *Bound* e da *Bound* a *Shadowed*.

Unbound → Bound

Quando un PA si connette con successo al canale del PT e inizia a ricevere dati, può inviare la PAD generata al modello del DT. La generazione della PAD può essere utilizzata come passo di sincronizzazione per segnalare che il PA è connesso con successo al PT. Il modello del DT è quindi responsabile di raccogliere le diverse PAD, valutare se tutte le informazioni rilevanti per iniziare a calcolare lo stato del DT sono disponibili e, quindi, passare alla fase *Bound*.

Bound → Shadowed

Per permettere al DT di passare dallo stato Bound allo stato Shadowed, il modello definisce quali proprietà devono essere monitorate sul Physical Asset e inizia ad osservarle attraverso i PA. Gli step coinvolti sono:

1. il Model definisce quali proprietà monitorare e inizia ad osservarle;

2. i Physical Adapter coinvolti comunicano con il Physical Asset, ricevono dati e generano eventi per notificare cambiamenti nelle proprietà fisiche;

3. gli eventi ricevuti sono utilizzati dal Digital Twin Model per eseguire la Shadowing Function e computare il nuovo DT State;

4. il DT può passare dalla fase Bound a quella Shadowed fino a quando è in grado di mantenere una sincronizzazione appropriata con il Physical Asset nel tempo.

Questi meccanismi consentono di gestire il comportamento del DT in modo coerente, anche con la complessità aggiuntiva del design modulare della PI.

2.2.6 Digital Twins ecosystem

In parallelo all'evoluzione dei modelli architetturali dei singoli DT, è emersa nel 2022 una visione più ampia con il *Web of Digital Twins* (WoDT) [11]. Tale approccio estende il paradigma dei Digital Twin dalla virtualizzazione di asset isolati verso la creazione di ecosistemi distribuiti di DT interconnessi, in grado di rappresentare realtà fisiche complesse, dinamiche e cross-domain.

Gestione centralizzata L'architettura WoDT prevede componenti infrastrutturali esterni ai singoli DT che supportano la gestione dell'ecosistema. Il **Digital Twin Manager (DTM)** è responsabile della gestione del ciclo di vita dei DT, dalla creazione e assegnazione di identificatori univoci fino alla dismissione. Questo componente offre servizi per individuare DT basandosi su ID o proprietà specifiche, fungendo da registro centrale dell'ecosistema.

Altro aspetto fondamentale è che multipli Digital Twin possono essere eseguiti sullo stesso **Digital Twin Engine**, che orchestra l'esecuzione concorrente di diverse istanze DT attraverso un meccanismo multi-thread. Questo approccio consente di ottimizzare le risorse computazionali e facilita la comunicazione tra DT che operano nello stesso contesto, pur mantenendo l'indipendenza logica di ciascun gemello digitale.

Interoperabilità tra Digital Twins La comunicazione tra DT avviene attraverso le *Relationships* modellate esplicitamente nello stato di ciascun DT. Queste relazioni, rappresentate come link verso altri DT (tramite IRI univoci), permettono di navigare l'ecosistema e di accedere alle informazioni di DT correlati. Il **Distributed Knowledge Graph Engine (DKGE)** fornisce i mezzi per navigare l'intero grafo di DT interconnessi, instradando e inoltrando richieste di accesso ai dati verso i DT coinvolti in una query, senza richiedere conoscenza a priori della topologia dell'ecosistema.

Digital Interface condivisa Per favorire l'uso di DT come servizi e per facilitare l'interoperabilità degli ecosistemi, è possibile uniformare i digital adapter di diversi DT in un'unica Digital interface, la cui DTD descrive tutte le funzionalità e relazioni dell'ecosistema.

2.2.7 Sintesi architetturale

In Figura 2.2 il diagramma riassume l'architettura spiegata, unificando nello schema la suddivisione in moduli e le comunicazioni tra essi con l'idea di un ecosistema con 2 DT.

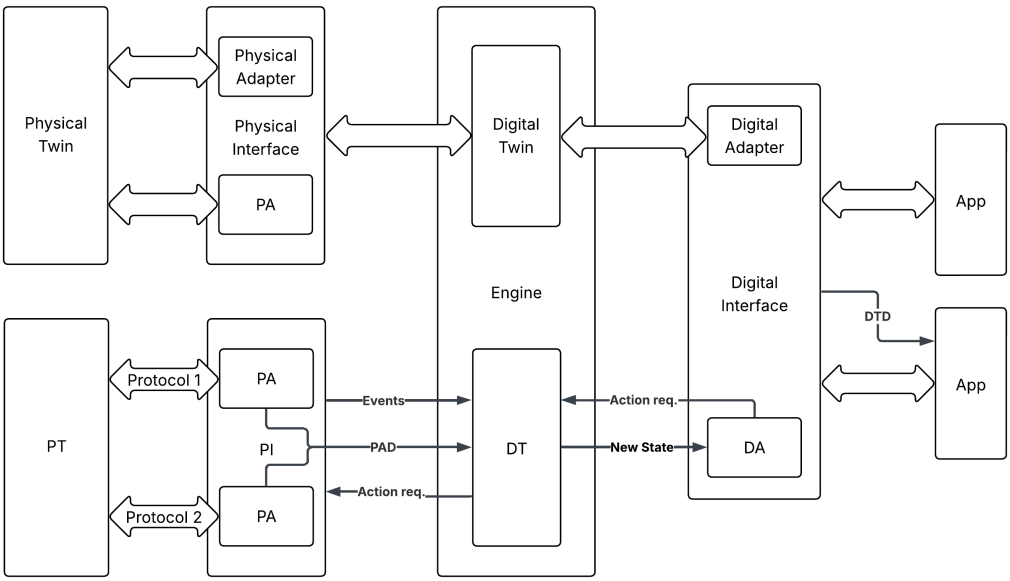

Figura 2.2: Diagramma riassuntivo

Questo modello fornisce una base concettuale solida per l'implementazione di Digital Twin che siano:

- **Modulari**: attraverso adapter specializzati a responsabilità singola;
- **Interoperabili**: capaci di adattarsi all'eterogeneità fisica e digitale;
- **Scalabili**: utilizzabili sia come entità standalone che come componenti di ecosistemi distribuiti;
- **Riusabili**: con componenti configurabili per diversi contesti applicativi.

Il framework WLDT (White Label Digital Twins), presentato nella sezione successiva, si propone come implementazione di questa architettura, fornendo una libreria che concretizza i principi discussi e ne facilita l'adozione in scenari applicativi reali.

2.3 WLDT

2.3.1 Introduzione

Il framework WLDT (White Label Digital Twins) è una libreria Java open-source progettata per semplificare lo sviluppo e l'implementazione di Digital Twins nell'ambito dell'Internet of Things [9]. L'obiettivo principale del framework è massimizzare modularità, riusabilità e flessibilità, permettendo di creare in modo efficace repliche digitali di oggetti fisici intelligenti. WLDT si propone come soluzione generale per la creazione di DT che possano essere eseguiti sia in cloud che su dispositivi edge, il design del framework è guidato da tre importanti requisiti:

- *Semplicità* – gli sviluppatori devono poter creare nuove istanze utilizzando moduli esistenti o personalizzando il comportamento secondo le esigenze dello scenario applicativo;

- *Estendibilità* – pur mantenendo un core semplice e leggero, le API devono essere facilmente estendibili per permettere la personalizzazione e l'aggiunta di nuove funzionalità attraverso il caricamento ed esecuzione di moduli multipli;

- *Portabilità e Microservice Readiness* – un Digital Twin implementato con WLDT deve poter essere eseguito su qualsiasi piattaforma senza modifiche, supportando lo sviluppo di applicazioni DT modellate come agenti software indipendenti e impacchettate come microservizi.

2.3.2 Implementazione dell'architettura

WLDT implementa l'architettura modulare attraverso una struttura stratificata su tre livelli principali (Figura 2.3).

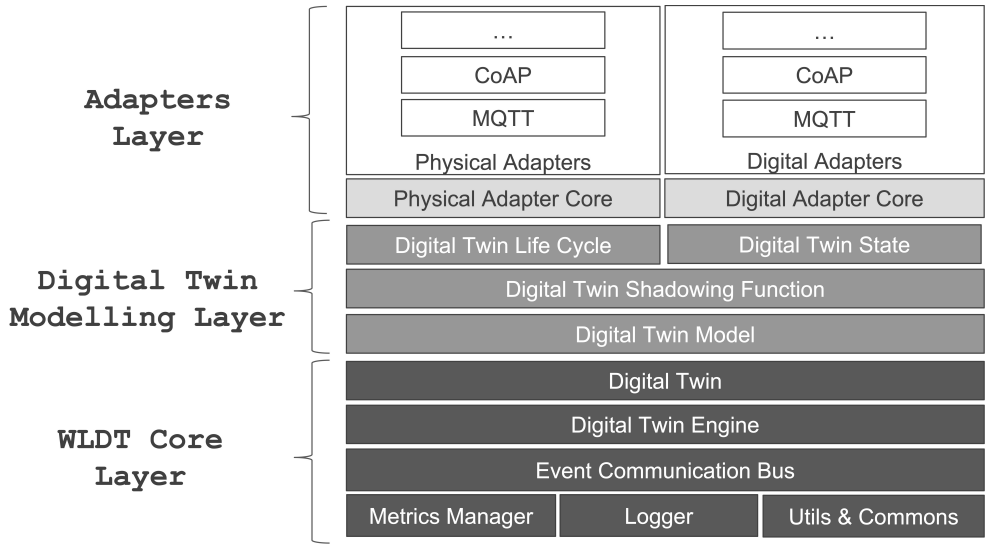

Figura 2.3: Schema dell'architettura a strati dalla documentazione WLDT [9]

Livello Core

Il livello core fornisce le funzionalità infrastrutturali specifiche dell'implementazione WLDT:

Digital Twin Engine : definisce il motore multi-thread della libreria, permettendo l'esecuzione e il monitoraggio di multipli DT simultaneamente. È responsabile dell'orchestrazione dei diversi moduli interni dell'architettura. Attualmente supporta l'esecuzione di gemelli nello stesso processo Java, realizzando concretamente il concetto di engine condiviso discusso nell'architettura WoDT. La stessa astrazione del motore potrebbe essere estesa per supportare l'esecuzione distribuita.

Event Communication Bus rappresenta una specifica implementazione event-driven della comunicazione interna al DT. Progettato per supportare la comunicazione tra i diversi componenti dell'istanza DT, permette di definire eventi personalizzati per modellare input e output fisici e digitali. Ogni componente WLDT può pubblicare sul bus condiviso e definire un Event Filter per specificare quali tipi di eventi gestire, associando callback specifici per elaborare i diversi messaggi.

Storage Layer : integrato nel core con l'obiettivo di abilitare un salvataggio manuale o automatico dei dati relativi all'evoluzione dello stato dei Digital Twins, degli eventi generati e processati, e di qualsiasi variazione che coinvolge

proprietà, eventi, azioni, relazioni e ciclo di vita. Questa funzionalità è specifica dell'implementazione WLDT e non fa parte dell'architettura astratta.

Livello di Modellazione

Il livello di modellazione implementa i concetti architetturali di Model e Shadowing Function:

Digital Twin State : usa il modello $S_{DT} = \langle P, R, E, A \rangle$ definita nell'architettura, gestendo la lista di proprietà, eventi, azioni e relazioni. Ogni modifica genera automaticamente un evento di notifica per tutti i componenti interessati.

Shadowing Function : componente fondamentale che deve essere esteso dal designer del DT per concretizzare il modello specifico. Implementa le funzioni di shadowing $Shad_{PT \to DT}$ e $Shad_{DT \to PT}$ attraverso un'architettura basata su callback. La Shadowing Function osserva il ciclo di vita del Digital Twin ed è notificata dei cambiamenti di stato, ad esempio quando il DT entra nello stato Bound dopo che i Physical Adapter hanno completato la procedura di binding.

Livello Adapter

WLDT fornisce classi astratte per Physical e Digital Adapter che devono essere estese per protocolli specifici:

Physical Adapter : classe astratta che definisce le funzionalità essenziali per implementazioni specifiche. Ogni adapter produce la propria PAD attraverso metodi dedicati. Un DT può essere equipaggiato con multipli Physical Adapter, e transita dallo stato Unbound allo stato Bound quando tutti hanno prodotto le rispettive PAD.

Digital Adapter : fornisce callback che ogni implementazione specifica utilizza per essere notificata dei cambiamenti nello stato del DT. Simmetricamente ai Physical Adapter, un DT può definire multipli Digital Adapter per esporre stato e funzionalità attraverso diversi canali e protocolli.

2.3.3 Sistema Event-Driven

L'approccio event-driven di WLDT rappresenta una scelta implementativa specifica che disaccoppia i componenti attraverso un efficace scambio di messaggi. Il framework definisce due categorie principali di eventi:

Eventi del Physical Asset, che mappano interazioni bidirezionali con il PT:

- Physical Asset Description – variazioni nella PAD;
- Physical Asset Variation – variazioni di proprietà, eventi e relazioni;
- Physical Asset Action Request – richieste di azione verso il mondo fisico.

Eventi del Digital Twin, che mappano l'evoluzione del DT:

- Life Cycle Variation – cambiamenti di stato nel ciclo di vita;
- DT State Variation – variazioni dello stato del DT;
- DT State Event Notification – eventi generati dal DT;
- Digital Action Request – richieste di azione da applicazioni esterne.

2.3.4 Sviluppo di un Digital Twin con WLDT

Per implementare un Digital Twin utilizzando WLDT, è necessario:

1. Estendere la classe astratta `ShadowingFunction` per definire il modello specifico e le logiche di shadowing;
2. Implementare almeno un Physical Adapter estendendo `PhysicalAdapter` per il protocollo di comunicazione con il PT;
3. Implementare almeno un Digital Adapter estendendo `DigitalAdapter` per esporre il DT alle applicazioni;
4. Istanziare il `WldtEngine` e registrare il DT con i suoi adapter;
5. Avviare l'engine per iniziare il ciclo di vita del DT.

La libreria WLDT fornisce già diverse implementazioni di adapter pronte all'uso che richiedono solamente una configurazione appropriata per il contesto applicativo specifico. Tra questi, sono disponibili:

- `MqttPhysicalAdapter` e `MqttDigitalAdapter`: per comunicazione tramite protocollo MQTT, ampiamente utilizzato in scenari IoT;
- `HttpDigitalAdapter`: per esporre il DT attraverso API REST standard;
- `WoTPhysicalAdapter` e `WoTDigitalAdapter`: adapter conformi allo standard Web of Things del W3C, che costituiscono il contributo principale di questo lavoro e saranno approfonditi nei capitoli successivi.

Capitolo 3

WoT Adapters: requisiti e funzionalità

3.1 Integrazione dei Digital Twin nel Web of Things

L'integrazione tra Digital Twin e Web of Things rappresenta una delle evoluzioni più promettenti per la realizzazione di ecosistemi cyber-fisici interoperabili, scalabili e semantici. Sebbene i due paradigmi siano stati concepiti con obiettivi distinti — il WoT per l'interoperabilità tra dispositivi eterogenei e il DT per la rappresentazione dinamica e sincronizzata di entità fisiche — la loro convergenza consente di superare molte delle attuali limitazioni dei sistemi IoT tradizionali.

3.1.1 Confronto tra Thing WoT e Digital Twin

Prima di procedere con l'analisi e l'attuazione dell'integrazione, è utile comprendere le analogie strutturali e le differenze concettuali tra una Thing WoT e un Digital Twin.

Entrambi hanno l'obiettivo di rappresentare entità fisiche in formato digitale accessibile, fornendo meccanismi per il monitoraggio e il controllo remoto. Per raggiungere questo scopo nel modo più conciso ed efficace possibile, il modello della Thing Description e quello dello stato del Digital Twin condividono un'organizzazione strutturale simile. Entrambi definiscono proprietà come attributi leggibili che rappresentano lo stato dell'entità, azioni per rappresentare operazioni invocabili che possono produrre effetti nel dispositivo che rispecchiano e gli eventi come notifiche asincrone generate in risposta a cambiamenti di stato o condizioni specifiche.

Nonostante queste analogie strutturali, comuni nella rappresentazione di dispositivi IoT, esistono differenze sostanziali che derivano dalle finalità distinte dei due modelli. La più significativa è data dalla capacità del Digital Twin, attraverso il processo di shadowing, di effettuare computazione autonoma, producendo dati elaborati e aggiuntivi rispetto a quelli del Physical Asset. Questo gli può permettere anche di continuare a simulare il Physical Twin quando esso non è raggiungibile. Una Thing WoT, diversamente, è pensata come una rappresentazione passiva di interfaccia che espone lo stato corrente dell'entità fisica o digitale che rappresenta, senza elaborazioni intermedie. Ciò porta con sé un'altra differenza architetturale: il Digital Twin ha necessità di definire un ciclo di vita esplicito con determintati stati che regolano la sincronizzazione con il Physical Asset, mentre una Thing WoT è tipicamente sempre attiva finché i server che la espongono sono raggiungibili.

3.1.2 Due direzioni di integrazione

Analizzando i casi d'uso descritti nei capitoli precedenti, emerge come le applicazioni tipiche del WoT si sovrappongano significativamente ai domini dei Digital Twin. Si possono così identificare due strategie di integrazione complementari, ciascuna con obiettivi e modalità di implementazione distinte che rispondono a esigenze specifiche dei sistemi cyber-fisici moderni.

La Thing come Physical Asset per un DT

Nella prima soluzione, la *Thing* WoT rappresenta l'*asset* fisico di riferimento per la costruzione del Digital Twin. Le specifiche del W3C WoT, in particolare la *Thing Description* (TD), forniscono una rappresentazione unificata e indipendente dalla tecnologia di comunicazione sottostante, permettendo di accedere ai dati e alle funzionalità di dispositivi eterogenei attraverso interfacce web standardizzate.

Questa soluzione risulta particolarmente efficace nei contesti complessi e dinamici, come le *smart city*, dove la varietà di sensori, attuatori e piattaforme rende difficile l'integrazione diretta delle sorgenti informative. In tale contesto, ogni dispositivo o sottosistema urbano può essere modellato da una Thing Description, la quale permetterebbe la creazione automatica di una PAD per il DT della città. Attraverso gli standard WoT, i dati reali possono essere acquisiti, interpretati e integrati all'interno di Digital Twin complessi, che rappresentano in modo aggiornato e sincronizzato lo stato della città. In questa prospettiva, il WoT non solo fornisce un'infrastruttura uniforme per la connessione degli oggetti fisici, ma diventa la base semantica per la costruzione automatizzata di gemelli digitali di sistemi complessi, come quelli urbani.

Il Digital Twin esposto come Thing WoT

La seconda direzione di integrazione considera invece il *Digital Twin* come un'entità digitale che espone le proprie capacità e funzionalità attraverso le tecnologie del Web of Things. In questo caso, la *Thing Description* funge da DTD, permettendo ad applicazioni esterne di scoprire, comprendere e interagire con i servizi che essa offre, basandosi su implementazioni standard di Consumer WoT. Questo scenario è proposto anche nella documentazione ufficiale del W3C stesso, tra i principali pattern di deployment dell'architettura WoT. Nel dominio industriale, ad esempio, i diversi componenti fisici di un processo produttivo possono essere rappresentati da Digital Twin che riproducono il loro comportamento e ne espongono le funzioni operative. L'uso delle TD come Digital Twin Description consente di descrivere tali DT in modo standardizzato, rendendoli facilmente accessibili a piattaforme di analisi, simulazione o ottimizzazione di livello superiore.

In sintesi, le due direzioni di integrazione — la *Thing* come asset fisico e il DT come Thing WoT — delineano un ciclo continuo di connessione tra mondo reale e rappresentazione virtuale, fondato su principi di interoperabilità, scalabilità e riusabilità. Questa duplice relazione conferma il ruolo del Web of Things come infrastruttura chiave per l'evoluzione dei sistemi basati su Digital Twin.

3.1.3 Realizzazione tramite adapters

L'architettura modulare di WLDT permette di concretizzare le due modalità di integrazione descritte estendendo rispettivamente le classi di Physical Adapter e Digital Adapter.

Ruolo del PA Utilizzare una Thing WoT conforme allo standard W3C come Physical Asset per la costruzione del Digital Twin. Il WoT Physical Adapter agisce come Consumer della Thing, acquisendo e parsando la TD per generare automaticamente la Physical Asset Description. L'adapter si occupa di leggere le proprietà, sottoscriversi agli eventi e invocare le azioni esposte dalla Thing, gestendo autonomamente la comunicazione attraverso i Protocol Binding specificati nei forms e traducendo le interazioni WoT in eventi WLDT comprensibili dalla Shadowing Function.

Ruolo del DA Rendere il Digital Twin accessibile all'esterno attraverso una Digital Interface conforme allo standard WoT. Il WoT Digital Adapter agisce come Exposer del Digital Twin, osservando lo stato corrente del DT e generando dinamicamente una Thing Description che lo rappresenti fedelmente.

L'adapter implementa i Protocol Binding necessari per esporre le affordances e traduce le richieste WoT in eventi WLDT comprensibili dal modello. Le applicazioni esterne possono così interagire con il DT utilizzando i protocolli e le convenzioni del WoT, mentre il Digital Twin mantiene le sue capacità di shadowing e computazione autonoma, rendendole accessibili tramite l'interfaccia standardizzata.

3.1.4 Sintesi dei vantaggi

L'integrazione tra WLDT e WoT presenta quindi vantaggi significativi:

- **Standardizzazione**: le Thing Description forniscono un formato universalmente riconosciuto per descrivere le capacità dei Digital Twin.
- **Interoperabilità**: l'adesione agli standard WoT facilita l'integrazione di Digital Twin con ecosistemi IoT esistenti.
- **Scalabilità**: l'architettura modulare di WLDT può sfruttare l'infrastruttura WoT per i deployment distribuiti.
- **Riusabilità**: adapter WoT configurabili conformi allo standard possono essere utilizzati per tradurre qualsiasi Thing Description valida.

3.2 WoT Physical Adapter

Estendendo la classe astratta `ConfigurablePhysicalAdapter`, questo componente implementa la logica necessaria per consumare una Thing conforme allo standard W3C e tradurne le interazioni nel modello event-driven di WLDT. L'adapter si posiziona nel livello Physical Interface dell'architettura WLDT e opera come client nel paradigma WoT, assumendo il ruolo di Consumer rispetto alla Thing target.

3.2.1 Lifecycle

Il WoT Physical Adapter segue il ciclo di vita standard degli adapter WLDT, applicando negli stati le procedure necessarie al collegamento e scollegamento della Thing esposta con il Digital Twin a cui è agganciato.

Fase di Binding (Unbound → Bound)

Durante la fase di binding, l'adapter:

1. Acquisisce la Thing Description dalla sorgente configurata

2. Valida la TD rispetto allo schema W3C

3. Genera la Physical Asset Description mappando le affordances

4. Pubblica la PAD tramite il metodo `notifyPhysicalAdapterBound()`

Se l'acquisizione della TD fallisce (es. Thing non raggiungibile, TD malformata), l'adapter segnala l'errore e riprova periodicamente, solo quando avviene con successo avvia la transizione allo stato di Bound.

Fase di Sincronizzazione

Una volta in stato Bound, l'adapter inizia la sincronizzazione con il Physical Asset e avvia procedure per mantenerla:

1. Inizializza i client per i protocolli necessari

2. Legge i valori correnti di tutte le proprietà e li pubblica al DT

3. Attiva le sottoscrizioni per proprietà ed eventi observable

4. Avvia i task di polling per le proprietà non-observable

Fase di arresto (Bound → Unbound)

Quando il modello richiede l'arresto dell'adapter, tipicamente in preparazione allo shutdown del Digital Twin o per una riconfigurazione, si avvia una procedura ordinata di terminazione:

1. Annulla tutte le sottoscrizioni all'osservazione di eventi e proprietà osservabili

2. Esce dai cicli di polling per la lettura delle proprietà

3. Termina tutte le task concorrenti ancora in corso

4. Segnala l'arresto avvenuto tramite il metodo notifyPhysicalAdapterUnBound()

3.2.2 Inizializzazione e Binding

Acquisizione della TD

Per avviarsi l'adapter richiede la TD della Thing che rappresenta il Physical Asset. Questa è comunemente condivisa tramite un URL web, ma si fornisce anche la possibilità di caricarla da un percorso file nel caso in cui la TD sia stata acquisita precedentemente e salvata su disco, o di passare direttamente la stringa Json, quando è generata dinamicamente o incorporata nella configurazione dell'applicazione. La sorgente, indipendentemente dal tipo, può essere incapsulata all'interno della configurazione dell'adapter, fornita al momento della creazione.

Per poter passare allo stato Bound, il Physical Adapter deve fornire la Physical Asset Description al modello del DT. Poiché tutta l'informazione necessaria è contenuta nella Thing Description, questa può essere immediatamente tradotta in una PAD. Sebbene i due formati siano concettualmente simili, come discusso in precedenza, esistono differenze di espressività e semantica che richiedono scelte implementative appropriate durante la mappatura.

Mappatura delle Affordances

La mappatura segue un approccio uno-a-uno per le tre categorie di affordance:

Properties Per ogni Property della TD si mappa una proprietà della PAD specificando:

- Il nome, (corrispondente alla chiave nel dizionario properties)
- Il valore iniziale
- Se è mutabile
- Se è scrivibile

La PAD WLDT non richiede di specificare esplicitamente il tipo o il relativo schema, poiché questi vengono dedotti dal valore iniziale, che, per questa ragione, diventa in essa un campo obbligatorio. Nella TD, diversamente, il valore default è opzionale: quando non è presente, l'adapter deve creare appositi valori iniziali che rispecchino lo schema del tipo specificato nella TD.

Actions Le azioni vengono mappate preservando:

- Il nome dell'azione
- il tipo semantico che può essere specificato nel campo `@type` della TD
- il tipo di payload dell'input derivato dallo schema input della action

È importante notare che nella TD è possibile indicare anche uno schema per l'output dell'azione, ma il modello di WLDT non prevede questo concetto, assumendo invece che un eventuale cambiamento di stato provocato dall'azione venga trasmesso attraverso cambiamenti di proprietà osservabili. Questa differenza riflette filosofie diverse: il WoT permette azioni con side effects espliciti comunicati tramite il valore di ritorno, mentre WLDT preferisce un modello più orientato allo stato dove tutti i cambiamenti si riflettono nelle proprietà.

Events Le caratteristiche degli eventi tradotti sono:

- Il nome dell'evento
- il tipo di payload, basato sullo schema nella TD

Limitazioni nella mappatura

Oltre alle differenze già evidenziate nella mappatura delle singole affordances, i metadati generali presenti nella TD non trovano una corrispondenza diretta nella PAD. Informazioni come title e description, che nella TD forniscono documentazione human-readable della Thing, non hanno un equivalente nella PAD che è pensata per uso interno machine-to-machine. Le informazioni di sicurezza globali, specificate nelle securityDefinitions della TD, non vengono trasferite nella PAD poiché la gestione della sicurezza è demandata al livello di comunicazione sottostante gestito dal Servient. I link a documentazione esterna e le forme specifiche per operazioni avanzate presenti nella TD rappresentano metadati di livello superiore che non hanno rilevanza per il funzionamento interno del Digital Twin. Si nota quindi che la Thing Description ha una capacità espressiva maggiore rispetto alla Physical Asset Description, riflettendo il suo ruolo più ampio di documentazione completa per consumer esterni rispetto alla finalità più focalizzata della PAD.

3.2.3 Sincronizzazione

Aggiornamento delle Proprietà

Passato allo stato Bound, il Physical Adapter deve mantenere lo stato del Digital Twin sincronizzato con il Physical Asset nel tempo. La prima operazio-

ne consiste nell'aggiornare le proprietà per sostituire i valori iniziali passati con la PAD, che potrebbero essere valori default generati automaticamente, con i valori effettivamente correnti letti dalla Thing. Successivamente è necessario avviare una procedura persistente che garantisca l'aggiornamento continuo dei valori per tutta la durata della vita del DT. Infatti, nel modello event-driven del Digital Twin, è il Physical Adapter che deve inviare proattivamente eventi di aggiornamento al DT quando lo stato del Physical Asset cambia. Nel paradigma WoT invece è il Consumer che decide quando leggere una proprietà, seguendo il paradigma request - response tipico del web. Solo se una proprietà è marcata come `observable` nella TD è possibile sottoscriversi a notifiche di cambiamento. Di conseguenza è l'adapter che deve implementare una procedura di lettura periodica delle properties, per garantire che lo stato sia aggiornato con ritardo trascurabile rispetto al dominio applicativo. Questo disallineamento richiede strategie differenti a seconda della natura della proprietà.

Proprietà Observable Se una proprietà espone un'operazione `observe` nelle sue Forms, l'adapter può sottoscriversi direttamente agli aggiornamenti implementando il pattern di osservabilità nativo del WoT. Questo rappresenta il caso ideale in quanto minimizza il traffico di rete, eliminando la necessità di polling periodico, e garantisce aggiornamenti tempestivi poiché le notifiche vengono inviate dal Physical Asset immediatamente quando il valore cambia. L'adapter gestisce la sottoscrizione secondo il protocollo specificato nella form corrispondente, che potrebbe utilizzare il subprotocol longpoll su HTTP, observe su CoAP, o subscription su topic MQTT.

Proprietà Non-Observable Per le proprietà che non supportano osservabilità, situazione comune in molti dispositivi legacy o con risorse limitate, l'adapter deve implementare un meccanismo di polling attivo. La frequenza di polling può essere configurata globalmente, applicando una singola frequenza a tutte le proprietà non-observable, oppure per-property, con frequenze differenziate in base all'importanza della proprietà o alla velocità di cambiamento attesa nel dominio specifico. Ad esempio, in un sistema di monitoraggio industriale, la temperatura di un processo critico potrebbe richiedere polling ogni secondo, mentre il livello di un serbatoio potrebbe essere letto ogni minuto. Poiché la TD non prevede nessun campo dedicato a esprimere periodi di lettura consigliati, i parametri di polling devono essere forniti esplicitamente nella configurazione dell'adapter, richiedendo conoscenza del dominio applicativo da parte di chi configura il sistema.

Propagazione eventi

Gli eventi esposti dalla Thing vengono gestiti attivando le relative sottoscrizioni secondo i Protocol Binding specificati nelle forms. Quando un evento viene ricevuto, l'adapter lo traduce in un `PhysicalAssetEventNotification` che viene pubblicato sul bus interno di WLDT, permettendo alla Shadowing Function di reagire appropriatamente, ad esempio aggiornando proprietà derivate o generando eventi di livello superiore nel Digital Twin State.

Invocazione di azioni

Il Physical Adapter espone anche un meccanismo per permettere al Digital Twin, tramite la Shadowing Function, di invocare azioni sul Physical Asset. Questo viene implementato nel metodo astratto `onPhysicalAssetActionRequest`, in risposta a eventi pubblicati sul bus interno di WLDT. Quando riceve una tale richiesta, l'adapter identifica l'affordance appropriata per l'azione nella Thing Description, costruisce il payload secondo lo schema input specificato, invoca l'azione utilizzando il protocollo e l'endpoint indicati nella form e restituisce un segnale di successo o fallimento.

Questo meccanismo bidirezionale, dove il DT può sia osservare che controllare il Physical Asset, è utile per implementare scenari di controllo in closed-loop dove il gemello digitale prende decisioni autonome basate sull'analisi dello stato e le attua direttamente sul sistema fisico.

3.2.4 Configurazione

Ricapitolando quanto detto la configurazione necessaria deve specificare:

- **Thing Description Source**: l'origine della TD, che può essere un URL web, un file locale o una stringa JSON.
- **Servient**: un Servient che implementa l'interfaccia del WoT, configurato con i protocol client necessari, in ordine di preferenza.
- **Polling Configuration**: per le proprietà non osservabili, la frequenza di polling desiderata, con possibilità di impostazione globale o di singola lettura.

3.3 WoT Digital Adapter

Estendendo la classe astratta `ConfigurableDigitalAdapter`, questo componente espone il Digital Twin a cui è agganciato come Thing WoT conforme allo standard W3C, rendendo lo stato e le capacità del DT accessibili a consumer esterni.

3.3.1 Lifecycle

Attivazione

Quando il Digital Twin segnala l'avvio al Digital Adapter si avvia la seguente procedura:

1. Notifica prima il binding con il DT attraverso l'invocazione del metodo appropriato nel framework WLDT, stabilendo la connessione logica con il modello.
2. Attende il sync con il DT e osserva il primo `DigitalTwinState`
3. Genera la Thing Description mappando affordances e metadati
4. Crea la Exposed Thing, aggiungendo gli handler per le varie operazioni
5. Pubblica la TD rendendola scopribile su un endpoint well-known, ed espone la Thing attraverso i server inizializzati, rendendola accessibile ai consumer esterni.

Out of sync

Quando il Digital Twin transisce in uno stato non sincronizzato, notifica l'evento al Digital Adapter. Il modello Web of Things però, non prevede un meccanismo per segnalare la perdita di sincronizzazione, l'implementazione deve quindi adottare una delle possibili strategie alternative. Una prima opzione consiste nell'arrestare completamente la exposed Thing e riattivarla quando il DT rientra nello stato Synchronized, ma in questo modo non si permette ai consumer di leggere gli ultimi valori validi delle proprietà prima della perdita di sincronizzazione, valori che potrebbero comunque essere utili per analisi o decision making. Una seconda opzione consiste nell'aggiungere una proprietà dedicata alla Thing Description, ad esempio un booleano denominato synchronized o inserire direttamente il timestamp contenuto nell'ultimo stato del DT ricevuto. I consumer possono così continuare ad accedere ai dati del DT essendo consapevoli del loro livello di aggiornamento e attendibilità, e decidere autonomamente se considerarli adeguati per il loro caso d'uso specifico.

Arresto

Quando viene richiesto l'arresto del Digital Twin, l'adapter esegue una procedura di shutdown ordinato simile a quella del Physical Adapter, terminando tutte le sottoscrizioni attive, arrestando i server di protocollo, e notificando lo stato di unbinding.

3.3.2 Generazione della Thing Description

Mappatura dello Stato del DT

Il processo di generazione della TD parte in seguito alla notifica di sincronizzazione del DT, basandosi sul primo `DigitalTwinState` che viene fornito all'invocazione del metodo e procede mappando ciascuna componente della sua struttura:

Properties Ogni proprietà dello stato viene mappata in una Property WoT con chiave corrispondente al nome. Nell'affordance si specifica lo schema del tipo, se readOnly o writeOnly e il valore default. Le proprietà del DT possono essere degli oggetti o delle collezioni, per cui è necessario inserire nella DT lo schema corrispondente alla classe dell'oggetto, descrivendola come struttura di oggetti e array.

Actions Le azioni vengono esposte come Actions WoT specificando il tipo semantico, il tipo dell'input, anch'esso secondo lo schema WoT, e utilizzando il nome come chiave.

Events Gli eventi diventano Events WoT con chiave il nome dell'evento e il tipo semantico specificato in type.

Relationships Il modello del Digital Twin include esplicitamente le relazioni che connettono il DT con altri Physical o Digital Twin, aspetto fondamentale nell'architettura WoDT per costruire ecosistemi di gemelli digitali interconnessi. Nel WoT, sebbene sia possibile referenziare altre Thing tramite link e creare strutture gerarchiche, questa connessione non può essere gestita direttamente dai singoli digital adapter dei DT delle componenti. La soluzione adottata consiste nel mappare le relationships come proprietà aggiuntive, aventi come chiave il nome, e come properties dello schema i campi della relazionie, tra cui un array per le Relationships Instances. In questo modo però, si può accedere ad esse solo leggendo il valore della properties mappata, e non si può dedurre nulla dalla sola TD, nonostante siano in parte una componente statica dello stato.

Froms e operazioni

Per ogni affordance generata è necessario aggiungere i rispettivi Form che specificano come accedere concretamente ad essa attraverso i protocolli disponibili. Il numero e il tipo di form dipendono dai server disponibili nel Servient fornito nella configurazione. Se il Servient ha configurato un server HTTP e un server MQTT connesso a un broker, l'adapter genererà due form per ciascuna affordance, con metodo e URL appropriati. Le operazioni disponibili in ciascun form possono essere derivate automaticamente da altre caratteristiche dell'affordance: una proprietà observable avrà le operazioni `observeproperty` e `unobserveproperty` in aggiunta a `readproperty`, mentre una proprietà readOnly non avrà l'operazione `writeproperty`.

Metadati Generali

I metadati generali della Thing Description vengono popolati come:

- (`@title` e `@description`) possono essere impostati dalla configurazione.
- Identificatore univoco della Thing (`id`), uguale all'id del Digital Adapter se non configurato diversamente.
- Il contesto semantico (`@context`, `@type`).

3.3.3 Sincronizzazione con lo Stato del DT

Ogni volta che la Shadowing Function elabora un cambiamento e aggiorna lo stato del Digital Twin, l'adapter viene notificato con il nuovo stato attraverso il meccanismo di eventi WLDT. La Thing esposta deve di conseguenza aggiornare le properties mappate per riflettere le correnti proprietà e relationship instances del DT. Per le proprietà marcate come observable nella TD, l'adapter deve anche inviare una notifica attiva a tutti i consumer che hanno sottoscritto l'observability di quella specifica proprietà, utilizzando il meccanismo specificato nel form.

Il modello del Digital Twin permette anche che venga aggiunto, rimosso o modificato strutturalmente un intero elemento dello stato durante l'evoluzione del DT. Ad esempio, un'azione che era sempre stata presente a partire dal primo stato di sincronizzazione potrebbe non trovarsi più in un aggiornamento successivo, perché non più disponibile a causa di un particolare evento rilevato sul Physical Asset, come un guasto hardware che disabilita una funzionalità o una riconfigurazione del sistema che ne modifica le capacità. Tale scenario implicherebbe una modifica nella Thing Description, che per continuare

a rispecchiare accuratamente le capacità del DT dovrebbe in questo caso rimuovere l'action corrispondente. Tuttavia, questo aspetto non è considerato dal modello Web of Things, il quale, prevedendo tipicamente una singola acquisizione iniziale della Thing Description, non dispone di meccanismi API standardizzati per modificare la TD e notificare ai consumer l'aggiornamento. La soluzione adottata, approfondita nel capitolo seguente, consiste nel creare ed esporre una nuova Thing Description, allo stesso end-point, in sostituzione a quella obsoleta.

3.3.4 Configurazione

Il WoT Digital Adapter richiede una configurazione che specifichi:

- **Metadati Generali**: informazioni come title, description, id.
- **Servient**: Servient che implementi l'API standard del W3C, configurato con i server sui quali si vuole esporre la Thing, in ordine di preferenza.
- **Osservabilità properties**: si specifica se si vogliono rendere le proprietà observable, selettivamente o nella totalità.

Capitolo 4

Sviluppo e soluzioni implementative

Questo capitolo si concentra sugli aspetti implementativi concreti degli adapter sviluppati. L'implementazione realizzata si propone di fornire un Physical Adapter e un Digital Adapter configurabili e riutilizzabili per il framework WLDT, che incapsulino l'integrazione automatica con gli standard W3C Web of Things, concretizzando i requisiti e le soluzioni discusse in prototipi software open-source.

4.1 Tecnologie

L'integrazione tra Digital Twin e Web of Things, discussa nel capitolo precedente, trova la sua realizzazione pratica nell'integrazione tra due framework JVM specifici: WLDT [9] per la gestione dei Digital Twin e Kotlin-WoT per l'implementazione delle specifiche del W3C. Questa sezione spiega le caratteristiche tecniche rilevanti del linguaggio di sviluppo e di Kotlin-WoT, essendo WLDT già stato approfondito nel capitolo 2.

4.1.1 Linguaggio Kotlin

Essendo un linguaggio che compila su Java Virtual Machine, Kotlin garantisce piena compatibilità binaria con il framework WLDT esistente, scritto in Java. Questa interoperabilità bidirezionale elimina la necessità di creare layer di integrazione complessi o wrapper per l'interazione tra i componenti, permettendo agli adapter sviluppati in Kotlin di essere utilizzati come normali classi Java in qualsiasi progetto basato su JVM.

Dal punto di vista della sicurezza del codice, Kotlin introduce un sistema di tipi che distingue esplicitamente tra riferimenti nullable e non-nullable a livello

di linguaggio. Questa caratteristica, assente in Java, consente di prevenire a compile-time errori runtime legati ai riferimenti nulli.

L'ecosistema di librerie standard di Kotlin offre costrutti nativi particolarmente rilevanti per l'implementazione degli adapter. Il supporto alle coroutine, integrato nella libreria standard del linguaggio, facilita la programmazione asincrona e concorrente, necessaria per gestire le comunicazioni di rete con le Thing, i protocolli di publish-subscribe e task persistenti come il polling delle proprietà. A differenza dei thread tradizionali di Java, le coroutine rappresentano un'astrazione più leggera e componibile, permettendo di scrivere codice asincrono con una sintassi sequenziale che ne migliora significativamente la leggibilità e la manutenibilità. Un ulteriore aspetto rilevante è il supporto alla reflection fornito dalla libreria `kotlin-reflect`. Questa funzionalità è stata impiegata nella costruzione automatica degli schemi delle classi per le proprietà del Digital Twin State, permettendo di ispezionare a runtime la struttura degli oggetti Kotlin e di generare dinamicamente le corrispondenti rappresentazioni schema conformi alla specifica W3C.

4.1.2 Kotlin-WoT

La realizzazione degli adapter si basa sul framework Kotlin-WoT, l'implementazione ufficiale per JVM delle specifiche W3C Web of Things sviluppata principalmente da Robert Winkler e inserita nel progetto Eclipse ThingWeb. Questa libreria fornisce un'implementazione completa delle WoT Scripting API definite dal W3C, insieme ai principali Protocol Binding specificati dallo standard, offrendo le primitive necessarie per creare sia Consumer che Exposer di Thing conformi.

L'elemento centrale della libreria è la classe `Servient`, che incapsula tutta la logica di gestione dei protocolli di comunicazione secondo l'architettura del W3C. Esso viene utilizzato per costruire oggetti di interfaccia `Wot`, la quale fornisce metodi come `produce` e `consume`.

Durante lo sviluppo degli adapter sono emerse alcune limitazioni della libreria Kotlin-WoT, dovute principalmente al fatto che il progetto è ancora in fase di sviluppo attivo e alcune funzionalità non sono completamente implementate. In particolare, è stata riscontrata la necessità di gestire manualmente l'avvio dei protocol client per alcuni protocolli, comportamento che dovrebbe essere automatizzato dal Servient secondo la specifica. Inoltre, la corretta gestione di alcune operazioni di observability è ancora limitata solo ad alcuni protocolli, come MQTT, mentre si è rivelata non ancora pienamente supportata per HTTP, che non include nativamente meccanismi di publish-subscribe.

4.2 WoT Physical Adapter

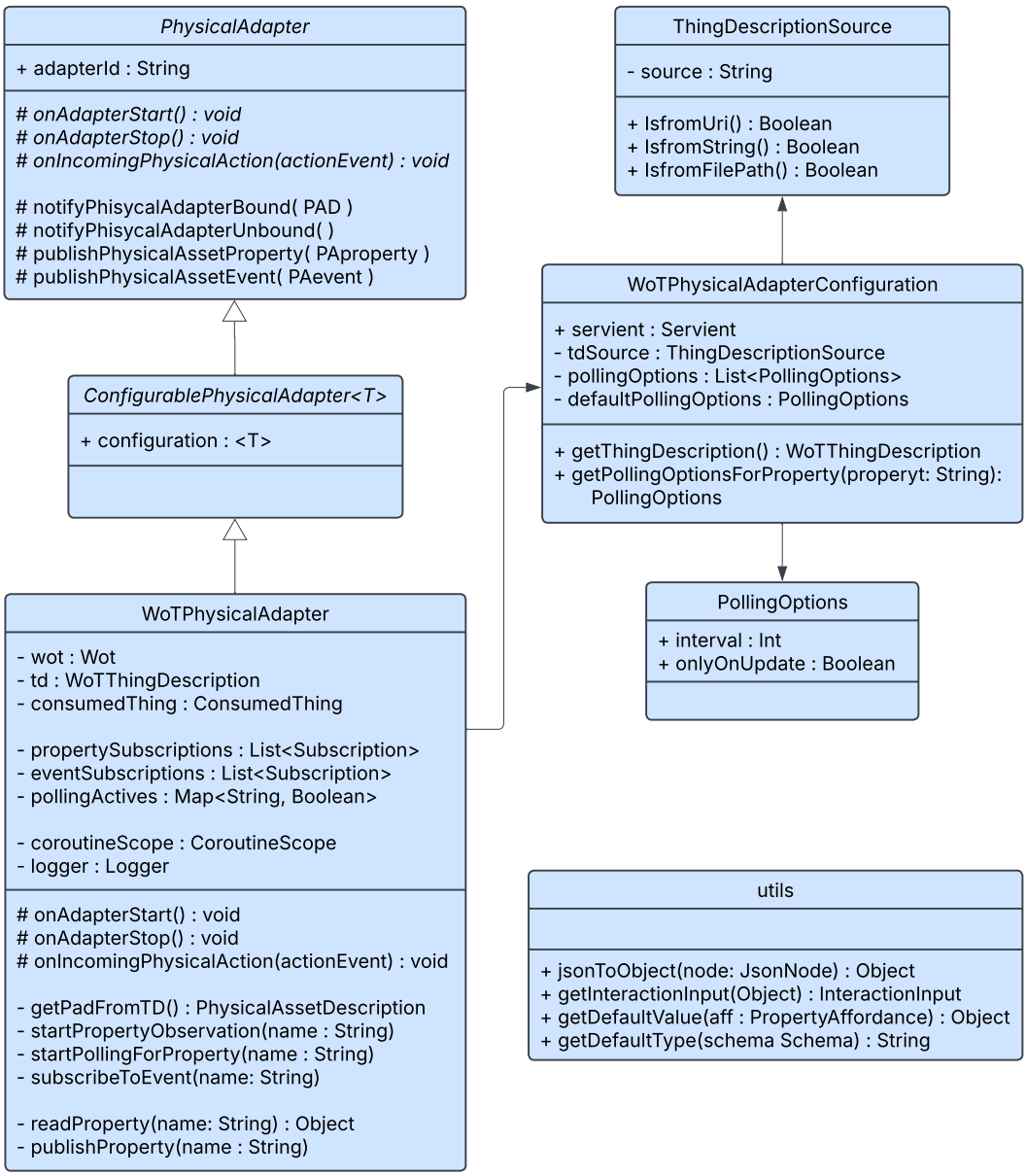

Figura 4.1: architettura WotPhysicalAdapter

La classe principale `WoTPhysicalAdapter` implementa la classe astratta `ConfigurablePhysicalAdapter<WoTPhysicalAdapterConfiguration>`, un'estensione di `PhysicalAdapter` che richiede una configurazione generica nel costruttore. Per queste due classi astratte di WLDT [9] sono stati riporta-

ti nel diagramma 4.1 specificamente solo i parametri e i metodi utilizzati o sovrascritti nel progetto.

4.2.1 Procedura di avvio

```
override fun onAdapterStart() {
  coroutineScope.launch {
    var bound = false
    while (!bound) {
      try {
        td = configuration.getThingDescription()
        val pad = getPadFromThingDescription(td)
        notifyPhysicalAdapterBound(pad)
        bound = true
      } catch (e: Exception) {
        logger.warn("Error obtaining Thing Description: ${e.message}.
            Retrying...")
        delay(1000)
      }
    }
    consumedThing = wot.consume(td)
    startClients()
    td.properties.forEach { (name, propertyAffordance) ->
      if (propertyAffordance.const == null ||
          propertyAffordance.const is NullSchema) {
        launch { publishProperty(name, readProperty(name)) }
        launch { startPropertyObservation(name, propertyAffordance) }
      } else {
        publishProperty(name, propertyAffordance.const!!)
      }
    }
    td.events.forEach { (name, _) ->
        launch { subscribeToEvent(name) }
    }
  }
}
```

Listato 4.1: Override del metodo astratto onAdapterStart()

Il loop di retry garantisce robustezza in scenari dove la Thing potrebbe non essere immediatamente disponibile, ad esempio durante l'avvio simultaneo di più componenti del sistema. L'uso di `delay()` evita busy-waiting e riduce il carico sulla rete. Una volta acquisita la TD e notificato lo stato Bound, l'adap-

ter procede con l'inizializzazione della consumed thing e l'avvio delle procedure di sincronizzazione. Le proprietà con valore costante (campo const popolato) vengono pubblicate immediatamente senza necessità di lettura remota, essendo il loro valore presente nella TD e non mutabile, mentre per le altre, si procede con lettura iniziale e attivazione della sincronizzazione continua. Infine, ma sempre parallelamente, si richiedono le sottoscrioni alle notifiche di eventi. Per l'invocazione di azioni invece, che provengono da DT, si è implementato l'override del metodo astratto `onIncomingPhysicalAction(action)`, chiamando `invocheAction` dalla consumed thing, dopo controlli e traduzione dell'input.

Dettagli generazione PAD

```
private fun getPadFromThingDescription(td: WoTThingDescription):
    PhysicalAssetDescription {
   return PhysicalAssetDescription().apply {
      td.properties.forEach { (name, propertyAffordance) ->
         val initialValue = getDefaultValue(propertyAffordance)
         val isImmutable = propertyAffordance.const != null
         val isWritable = !propertyAffordance.readOnly
         properties.add(PhysicalAssetProperty(name, initialValue,
             isImmutable, isWritable))
      }

      td.events.forEach { (name, eventAffordance) ->
            events.add(PhysicalAssetEvent(name,
                eventAffordance.objectType?.defaultType))
      }

      td.actions.forEach { (name, actionAffordance) ->
         val type = actionAffordance.objectType?.defaultType ?:
             "default"
         val input = actionAffordance.input
         val inputType = input?.objectType?.defaultType ?:
             getDefaultType(input)
         actions.add(PhysicalAssetAction(name, type, inputType))
      }
   }
}
```

Listato 4.2: Funzione per il mapping della TD in Physical Asset Description

La mappatura segue un approccio dichiarativo, dove per ciascuna categoria di affordance viene applicata una trasformazione specifica che preserva la semantica dell'elemento pur adattandola al modello WLDT. Per il calcolo del valore iniziale delle proprietà si implementa una strategia a tre livelli di priorità: valore costante (campo const), valore di default esplicito (campo default) e, se entrambi sono vuoti, si inserisce un valore neutro coerente con lo schema.

4.2.2 Gestione aggiornamento delle proprietà

```
private suspend fun startPropertyObservation(name: String,
    propertyAffordance: PropertyAffordance<*>) {
  val observationSuccessful = propertyAffordance.observable && try {
    val subscription = consumedThing.observeProperty(
      name,
      listener = { output ->
        publishProperty(name, jsonToJavaMapper(output.value()))
      },
      errorListener = { error ->
        //First Exception stops the observation
        startPollingForProperty(name)
      },
    )
    propertySubscriptions[name] = subscription
    subscription.active
  } catch (e: Exception) {
    false
  }
  if (!observationSuccessful) {
    startPollingForProperty(name)
  }
}
```

Listato 4.3: Procedura per attivare l'osservazione di una proprietà

Per le proprietà che espongono un'operazione `observeproperty` nelle loro Forms, l'adapter tenta di stabilire una sottoscrizione diretta utilizzando il pattern nativo WoT. La gestione della sottoscrizione include meccanismi di error handling che, in caso di fallimento dell'osservazione, attivano automaticamente un fallback al polling. Lo stesso approccio di subscribtion con listener è usato per la cattura degli events, che vengono notificati al DT dopo un'opportuna deserializzazione del Payload.

```
private fun startPollingForProperty(name: String) {
  val options = adapterConfig.getPollingOptionsForProperty(name)
  val interval = options.pollingInterval
  if (interval < 0) {
      return
  }

  coroutineScope.launch {
      logger.info("Starting polling for property '$name'")
      pollingActives[name] = true
      delay(interval)
      if (options.onlyUpdatedValues) {
        var lastValue : Any? = null
        while (pollingActives[name] == true) {
          val newValue = readProperty(name)
          if (newValue != lastValue) {
            lastValue = newValue
            publishProperty(name, newValue)
          }
          delay(interval)
        }
    } else {
      while (pollingActives[name] == true) {
        publishProperty(name, readProperty(name))
        delay(interval)
      }
    }
  }
}
```

Listato 4.4: Procedura per avviare il polling di una proprietà

L'implementazione del polling sfrutta le coroutine per eseguire un ciclo di letture periodiche in modo non bloccante . Ciascuna proprietà viene gestita da una coroutine dedicata che, utilizzando la funzione `delay`, si sospende ciclicamente per l'intervallo di polling configurato. Un parametro booleano della configurazione di polling permette inoltre di specificare se pubblicare ogni valore letto o solo i valori che differiscono dal precedente, ottimizzazione utile per ridurre il carico sul bus eventi WLDT quando le proprietà cambiano raramente. Le guardie dei cicli controllano uno specifico elemento di un vettore di booleani, campo della classe, in questo modo possono essere terminati naturalmente dalla procedura di arresto dell'adapter.

4.3 WoT Digital Adapter

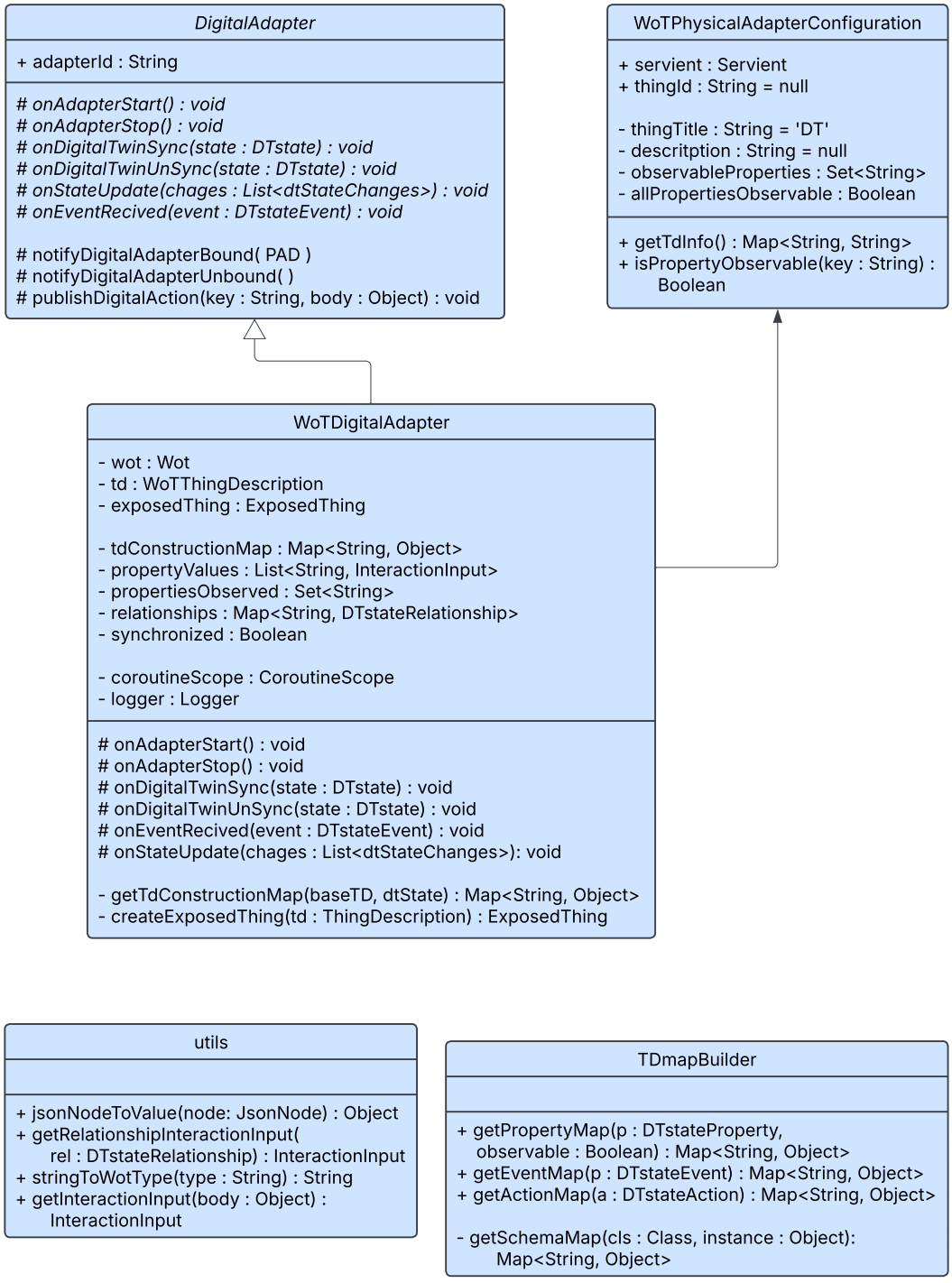

Figura 4.2: Architettura wot-digital-adapter

4.3.1 Avvio e sincronizzazione

Analogamente al `WoTPhysicalAdapter`, il `WoTDigitalAdapter` sovrascrive i metodi astratti della classe `DigitalAdapter` di WLDT, che estende. Similmente a quanto visto per il PA (listato 4.1), nel metodo `onAdapterStart()` si applica un ciclo di retry In questo caso però, esso controlla lo stato di sincronizzazione per assicurarsi che il campo `exposedThing` sia stato inizializzato, procedendo così a esporre la Thing, grazie al Servient fornito in configurazione. È quindi nella procedura `onDigitalTwinSync()` che si svologono le operazioni chiave per produrre l'exposed thing, che costituisce la rappresentazione del DT secondo il formato della Thing WoT.

```
override fun onDigitalTwinSync(digitalTwinState: DigitalTwinState?) {
  try {
    tdConstructionMap = getTdConstructionMap(
      configuration.tdInfo,
      digitalTwinState
    )
    val tempTD = ThingDescription.fromMap(tdConstructionMap)
    exposedThing = createExposedThing(tempTD)
    td = exposedThing.getThingDescription()

  } catch (e: Exception) {
    logger.error("Error creating Thing Description or Exposed Thing:
        ${e.message}", e)
  }
  synchronized = true
}
```

Listato 4.5: override del metodo astratto onDigitalTwinSync()

Creazione della TD e dell' Exposed thing

La documentazione di Kotlin-WoT suggerisce come approccio preferenziale per la creazione di un'istanza `ExposedThing` l'utilizzo di classi annotate che rappresentino il dispositivo da esporre, con metodi annotati per le specifiche interazioni. Questo pattern, tuttavia, presuppone che la struttura del dispositivo sia nota a compile-time, condizione non soddisfatta nel contesto di questo progetto. Una caratteristica fondamentale dell'adapter è infatti la capacità di esporre qualsiasi Digital Twin WLDT senza conoscerne preventivamente la struttura, derivando dinamicamente la Thing Description dallo stato del Digital Twin a runtime. È quindi necessario implementare la generazione di una TD temporanea di base mappando la struttura del primo

`DigitalTwinState`. Questa descrizione può essere così fornita alla funzione WoT standard `produce(ThingDescription)`, per ottenere l'exposed Thing (listato 4.7). Da essa infine, grazie alla libreria, si può ricavare automaticamente la TD completa, comprendente i Forms inseriti sulla base dei protocol server disopnibili nel servient (listato 4.5).

La generazione della TD temporanea si avvale della funzione `fromMap`, che consente di tradurre una struttura di mappe innestate in un oggetto `WoTThingDescription` valido. Questo approccio offre la garanzia di produrre una Thing Description conforme allo standard, poiché la validazione e la costruzione dell'oggetto sono delegate alla funzione di libreria. La struttura a mappe viene costruita aggregando una mappa dedicata per ciascuno dei quattro elementi dello stato del DT. Come discusso nel capitolo precedente, le relationships vengono inserite nella mappa delle proprietà, e il loro schema è ottenuto in maniera statica, essendo la struttura delle relazioni predeterminata dal modello WLDT. Si mostra di seguito la funzione per tradurre una proprietà nella rispettiva mappa dell'affordance.

```
internal fun getPropertyMap(p: DigitalTwinStateProperty<*>,
    observable : Boolean): Map<String, Any?> {
   val map = mutableMapOf<String, Any?>(
      "observable" to observable,
      "readOnly" to (!p.isWritable && p.isReadable),
      "writeOnly" to (!p.isReadable && p.isWritable),
      "default" to p.value,
   )
   map.putAll(getSchemaMap(p.value.javaClass, p.value))
   map["type"] = stringToWotType(p.type)
   return map
}
```

Listato 4.6: Metodo per il mapping dell'affordance di una proprietà

Il valore corrente della proprietà viene incluso come default nella Thing Description, fornendo ai consumer un'indicazione dello stato iniziale della Thing. La mappa per lo schema del tipo invece, viene generata dalla funzione `getSchemaMap`, di seguito analizzata.

Costruzione dello schema per tipi complessi

La generazione degli schemi per le affordances della Thing Description presenta sfide implementative legate alle limitazioni dell'introspezione dei tipi a runtime in Java. Sebbene lo stato del Digital Twin mantenga per ogni proprietà un campo stringa `type`, questo non garantisce informazioni sufficienti per ricostruire uno schema completo, in quanto potrebbe contenere semplicemente il nome qualificato della classe senza dettagli sulla struttura interna. Per questo motivo, l'implementazione si affida primariamente al valore corrente della proprietà, utilizzando la reflection per ispezionarne la struttura effettiva.

La funzione `getSchemaMap` opera ricorsivamente sulla classe e sull'istanza del valore per costruire lo schema appropriato. Per i tipi primitivi la conversione è diretta, mentre per i tipi complessi vengono ispezionate le proprietà dell'oggetto attraverso la reflection di Kotlin e Java. Nel caso di mappe, lo schema viene costruito iterando sulle coppie chiave-valore presenti nell'istanza.

Le limitazioni principali emergono nella gestione dei tipi generici. Per le collezioni e gli array, l'erasure dei tipi in Java impedisce di determinare staticamente il tipo degli elementi contenuti. L'implementazione tenta di estrarre il primo elemento della collezione per dedurne il tipo, ma quando la collezione è vuota lo schema degli elementi non può essere specificato in modo dettagliato. Analogamente, per le azioni del Digital Twin non è possibile ricostruire con precisione il tipo dell'input, poiché non esiste un'istanza concreta da cui derivare lo schema attraverso reflection.

Questi compromessi implementativi risultano accettabili nel contesto operativo dell'adapter: al momento della sincronizzazione del Digital Twin, le proprietà contengono tipicamente valori rappresentativi dello stato corrente, permettendo la generazione di schemi sufficientemente accurati per la maggior parte degli scenari pratici.

Gestione delle operazioni sulla Thing

Una volta prodotta l'exposedThing, è necessario configurare gli handler per ciascuna operazione esposta dalle affordances (listato 4.7). L'adapter mantiene due strutture dati principali per coordinare l'interazione con i consumer: una mappa `propertiesValues` che conserva i valori correnti di tutte le proprietà, e un insieme `propertiesObserved` che traccia quali proprietà richiedono l'invocazione di `emitPropertyChange` quando aggiornate. Per ogni proprietà viene impostato un handler di lettura che restituisce il valore memorizzato nella mappa. Le proprietà marcate come osservabili ricevono inoltre handler per la gestione delle sottoscrizioni, che aggiungono o rimuovono la proprietà dall'insieme delle osservate. Gli handler delle azioni traducono le invocazioni ricevute dai consumer in eventi WLDT che vengono pubblicati sul bus inter-

no. Per la trasmissione degli eventi, invece, si sovrascrive il metodo astratto `onEventReceived`, dove l'evento viene tradotto e segnalato ai consumer attraverso la funzione dell'API WoT `emitEvent()`.

```
private fun createExposedThing(td: WoTThingDescription):
    WoTExposedThing {
  val exposedThing = wot.produce(td)
  td.properties.forEach { (pKey, pAff) ->
    exposedThing.setPropertyReadHandler(pKey) {_ ->
        propertiesValues[pKey]
    }
    if (pAff.observable) {
      exposedThing.setPropertyObserveHandler(pKey) { _ ->
        propertiesObserved.add(pKey)
        null
      }
      exposedThing.setPropertyUnobserveHandler(pKey) { _ ->
        propertiesObserved.remove(pKey)
        null
      }
    }
  }
  td.actions.keys.forEach { aKey ->
    exposedThing.setActionHandler(aKey) { input, _ ->
      publishDigitalActionWldtEvent(aKey,
          jsonNodeToValue(input.value()))
      getInteractionInput(input.value())
    }
  }
  td.events.keys.forEach { eKey ->
    exposedThing.setEventSubscribeHandler(eKey) { _ ->
      observeDigitalTwinEventNotification(eKey)
    }
    exposedThing.setEventUnsubscribeHandler(eKey) { _ ->
      unObserveDigitalTwinEventNotification(eKey)
    }
  }
  return exposedThing
}
```

Listato 4.7: Funzione per produrre l'Exposed Thing e settare gli handler

4.3.2 Gestione dell'aggiornamento di stato

Il metodo `onStateUpdate` viene invocato dal framework WLDT ogni volta che la Shadowing Function elabora un cambiamento nello stato del Digital Twin. Come suggerito dalla documentazione WLDT, l'implementazione fa affidamento sulla lista dei cambiamenti fornita dal framework, in modo da poter rimappare selettivamente solo i valori variati. La procedura analizza ciascun cambiamento classificandolo in base al tipo di risorsa coinvolta. L'aggiornamento dei valori delle proprietà o delle istanze delle relazioni (che sono trattate in maniera equivalente), comporta un semplice aggiornamento dei valori memorizzati nelle strutture dati dell'adapter. L'aggiunta, modifica o rimozione di un elemento dello stato invece, costituisce una modifica strutturale che necessita la rigenerazione della Thing Description. Dunque, per ogni aggiornamento di questo tipo, si modifica la mappa per la costruzione della TD.

Come discusso nel capitolo precedente, lo standard WoT non prevede meccanismi per modificare dinamicamente una Thing Description già pubblicata. Coerentemente con questa limitazione, Kotlin-WoT non consente di alterare la struttura di un'ExposedThing dopo la sua creazione. L'unica strategia disponibile consiste nel distruggere completamente la Thing esistente e crearne una nuova con la struttura aggiornata.

L'implementazione gestisce questa transizione attraverso una sequenza di operazioni che prima rimuove l'ExposedThing dal Servient, poi genera una nuova Thing Description dalla mappa di costruzione aggiornata, crea la corrispondente ExposedThing e infine la espone nuovamente. Mantenendo invariato l'identificatore della Thing, essa rimane accessibile allo stesso endpoint, minimizzando l'impatto sui consumer esterni.

Una limitazione dell'architettura del Servient Kotlin-WoT impedisce la produzione di una Thing con lo stesso identificatore di una prodotta in precedenza, rendendo necessario completare la rimozione della Thing obsoleta prima di poter aggiungere quella nuova. Questa sequenzialità comporta inevitabilmente un breve intervallo temporale durante il quale la Thing non risulta esposta e accessibile ai consumer, aspetto che i sistemi integrati devono considerare nella gestione della resilienza.

4.4 Testing

La validazione degli adapter sviluppati richiede la predisposizione di ambienti di test che permettano di verificare sia la correttezza sintattica delle rappresentazioni generate, sia il corretto funzionamento operativo dell'integrazione tra WLDT e il paradigma WoT. Considerata la natura distribuita e le caratteristiche concorrenti dei due progetti, la strategia di testing adottata si concentra sulla verifica end-to-end delle catene di integrazione, piuttosto che su test unitari isolati.

4.4.1 Testing del WoT Digital Adapter

Per il Digital Adapter, la configurazione di test richiede un Digital Twin WLDT con uno stato sufficientemente articolato da esercitare tutte le capacità di mappatura dell'adapter: proprietà di tipi primitivi e complessi, azioni con input strutturati, eventi con payload e relazioni tra Digital Twin. A questo si aggiunge la necessità di un'applicazione consumer che permetta di verificare la correttezza sintattica della Thing Description generata e di testare operativamente le interazioni con la Thing esposta.

Scenario e componenti di test

L'ambiente di test è stato costruito utilizzando il `DemoPhysicalAdapter` fornito dal framework WLDT, un componente fittizio che simula un sensore di temperatura con proprietà, eventi, azioni e relazioni. Questo adapter è stato modificato per estendende la copertura dei test oltre i casi base già implementati, ad esempio aggiungendo una proprietà di tipo non primitivo. Una `DemoShadowingFunction` triviale implementa la mappatura diretta tra ciascun componente della Physical Asset Description e gli elementi corrispondenti nello stato del Digital Twin, eliminando complessità aggiuntive che potrebbero oscurare eventuali problemi nell'adapter stesso. La verifica della corretta generazione della Thing Description è stata condotta attraverso l'ispezione del documento JSON esposto dall'adapter, accessibile tramite browser all'endpoint configurato. Questa ispezione ha permesso di validare la presenza e la correttezza di tutte le affordances.

Risultati del mapping dello stato in TD

I listati seguenti illustrano il processo di mappatura da una proprietà del Digital Twin State alla corrispondente affordance nella Thing Description generata. Il primo (4.8) mostra un'istanza di `DigitalTwinStateProperty` che rappresenta la temperatura misurata dal sensore simulato, con un valore di

tipo complesso contenente la temperatura effettiva, l'ora della misurazione e una lista di allarmi booleani. Il secondo (4.9) presenta la corrispondente Property WoT generata automaticamente dall'adapter. La mappatura preserva la semantica della proprietà originale, traducendo il tipo complesso in uno schema JSON articolato che descrive ricorsivamente la struttura dell'oggetto con i suoi campi. Le Forms generate dal Servient riflettono i protocolli configurati, in questo caso MQTT, con endpoint distinti per le operazioni di lettura/scrittura e per la gestione dell'osservabilità. Il valore corrente viene inserito nel campo `default`, fornendo ai consumer un'indicazione dello stato iniziale della Thing. Questa mappatura dimostra la capacità dell'adapter di gestire automaticamente tipi complessi, generando schemi conformi allo standard WoT senza richiedere configurazioni aggiuntive o annotazioni esplicite nel codice del Digital Twin.

```
DigitalTwinStateProperty(
    key = 'temperature-property-key',
    value = TemperaturePropertyObject(23.5, 14, listOf(false, false,
        false)),
    type='physical.TemperaturePropertyObject',
    readable=true,
    writable=true,
    exposed=true
)
```

Listato 4.8: Istanza di una proprietà nello stato del DT

```
{
  "properties": {
    "temperature-property-key": {
      "type": "object",
      "forms": [
        {
          "href": "mqtt://localhost:61890/temperature-sensor-0/
                   properties/temperature-property-key",
          "contentType": "application/json",
          "op": [ "readproperty", "writeproperty" ]
        },
        {
          "href": "mqtt://localhost:61890/temperature-sensor-0/
                   properties/temperature-property-key/observable",
          "contentType": "application/json",
          "op": [ "observeproperty", "unobserveproperty" ]
        }
      ],
      "observable": true,
      "properties": {
        "temperature": {
          "type": "number"
        },
        "hour": {
          "type": "integer"
        },
        "alarms": {
          "type": "array",
          "items": {
            "type": "boolean"
          }
        }
      },
      "default": {
        "temperature": 0.0,
        "hour": 0,
        "alarms": [ false, false, false ]
      }
    }
  }
}
```

Listato 4.9: Affordance della property mappata nella TD

4.4.2 Testing del WoT Physical Adapter

La verifica del corretto funzionamento del Physical Adapter necessita di tre componenti principali

- una Thing WoT esposta che implementi la varietà di affordances previste dallo standard: proprietà sia osservabili che non osservabili, con schemi che spaziano dai tipi primitivi a strutture complesse, azioni che accettino input tipizzati ed eventi che trasportino payload significativi.
- un Digital Twin WLDT configurato con una Shadowing Function isomorfa.
- un Digital Adapter che permetta di osservare se gli aggiornamenti di stato nel Digital Twin riflettano correttamente le variazioni della Thing sorgente e che consenta l'invocazione di azioni per verificare la propagazione bidirezionale.

Evoluzione della strategia di test

Nelle fasi iniziali dello sviluppo, l'adapter è stato validato collegandolo a una Thing esposta da un progetto Kotlin dedicato, implementata attraverso il pattern delle classi annotate supportato da Kotlin-WoT. Questa configurazione ha permesso di verificare isolatamente il comportamento del Physical Adapter rispetto a una Thing con struttura nota e controllabile.

Successivamente alla realizzazione del WoT Digital Adapter, la strategia di testing è stata aggiornata sfruttando la composabilità degli adapter. La Thing prodotta dal test del Digital Adapter è stata utilizzata come sorgente per il test del Physical Adapter, creando di fatto un Digital Twin di un Digital Twin. Il secondo gemello digitale è stato collegato a un `DemoDigitalAdapter` per l'osservazione dello stato risultante. Questa configurazione ha offerto vantaggi significativi nella validazione: non solo ha permesso di verificare l'aggiornamento consequenziale dello stato attraverso la catena di sincronizzazione tra i due Digital Twin, ma ha anche consentito di testare efficacemente le operazioni di invocazione delle azioni e di sottoscrizione all'osservabilità delle proprietà dal lato del Digital Adapter, aspetti operativi difficilmente verificabili attraverso semplici interazioni tramite browser o client HTTP generici.

Questa configurazione composita ha inoltre dimostrato concretamente l'interoperabilità completa tra i due adapter sviluppati, validando che una Thing esposta da un Digital Twin può essere consumata da un altro Digital Twin. L'analisi comparativa degli stati dei due Digital Twin ha rivelato che la differenza più significativa tra il primo e il secondo gemello riguarda le relationships: queste, mappate come proprietà nella Thing Description dal Digital

Adapter per le limitazioni del modello WoT discusse nel capitolo precedente, vengono ricostruite come semplici proprietà nel secondo Digital Twin dal Physical Adapter. Nonostante questa trasformazione, l'informazione semantica contenuta nelle relazioni viene preservata e rimane accessibile attraverso le operazioni standard sulle proprietà.

Conclusioni

Il lavoro presentato in questa tesi dimostra la validità concettuale e la fattibilità tecnica dell'integrazione tra il paradigma dei Digital Twin e lo standard Web of Things del W3C. La realizzazione degli adapter ha confermato che le Thing Description WoT possono fungere efficacemente da Physical Asset Description per Digital Twin, permettendo di consumare dispositivi IoT conformi allo standard W3C come asset fisici nel framework WLDT. Simmetricamente, i Digital Twin possono essere esposti come Thing WoT, rendendoli accessibili ai consumer esterni attraverso protocolli standardizzati. Questa bidirezionalità stabilisce un ponte operativo tra ecosistemi separati, aprendo possibilità applicative concrete in domini quali smart city, Industry 4.0 e sistemi cyber-fisici complessi.

Il processo di sviluppo ha tuttavia evidenziato alcune differenze architetturali tra i due paradigmi che richiedono compromessi implementativi. La più rilevante riguarda il modello delle relationships dei Digital Twin, che non trovano una corrispondenza diretta nello standard WoT. Sebbene la mappatura in proprietà preservi l'informazione funzionale, comporta una perdita della semantica relazionale esplicita, limitando la capacità di ricostruire automaticamente ecosistemi complessi di Digital Twin interconnessi a partire dalle sole Thing Description. Analogamente, le differenze nella gestione degli output delle azioni e nei meccanismi di modifica dinamica delle descrizioni richiedono strategie specifiche che, pur funzionali, introducono complessità aggiuntiva.

L'architettura modulare del framework WLDT, che separa nettamente le responsabilità di interfacciamento fisico, logica di shadowing ed esposizione digitale, si è rivelata particolarmente adatta all'integrazione con lo standard WoT. L'inserimento avviene naturalmente nell'architettura attraverso l'implementazione degli adapter astratti, confermando la validità delle scelte progettuali alla base del framework. Questa modularità suggerisce che pattern simili potrebbero essere applicati per integrare altri standard o protocolli emergenti nell'ambito dei Digital Twin.

Sviluppi futuri

Una direzione di ricerca promettente potrebbe riguardare la gestione automatica degli ecosistemi di Digital Twin interconnessi. Un'estensione riguarderebbe l'introduzione nella configurazione del WoT Physical Adapter della capacità di specificare relationships tra Digital Twin. Questa funzionalità abiliterebbe la costruzione di un componente orchestratore capace di processare Thing Description che rappresentino sistemi composti da multiple Thing interconnesse, istanziando automaticamente un ecosistema di Digital Twin con le appropriate relazioni. Tale componente analizzerebbe la topologia descritta dalle interconnessioni tra le Thing, creerebbe un WoT Physical Adapter per ciascuna Thing identificata, e configurerebbe le relationships nel framework WLDT per riflettere la struttura del sistema originale. Questo meccanismo permetterebbe di tradurre automaticamente architetture IoT complesse, descritte secondo lo standard WoT, in ecosistemi di Digital Twin interconnessi e operativi, facilitando significativamente il deployment di sistemi cyber-fisici articolati e la loro gestione attraverso il paradigma dei gemelli digitali.

Simmetricamente, un'estensione significativa all'integrazione consisterebbe nello sviluppo di meccanismi che, a livello di Digital Interface, interpretino le relationships di diversi DT per costruire automaticamente strutture gerarchiche di Thing multiple. Questo permetterebbe di esporre un sistema complesso di Digital Twin interconnessi come un grafo di Thing correlate, ciascuna con la propria Thing Description e le proprie affordances, collegate attraverso link espliciti. Tale rappresentazione distribuita migliorerebbe significativamente la scalabilità e la componibilità dei sistemi basati su Digital Twin, permettendo ai consumer di navigare la topologia e di interagire selettivamente con i componenti rilevanti.

In conclusione, gli adapter sviluppati costituiscono una base solida per ulteriori evoluzioni che ne migliorino l'usabilità e l'adozione, contribuendo alla visione di ecosistemi cyber-fisici aperti e interoperabili basati su standard consolidati.

Bibliografia

[1] John Barton and Tim Kindberg. The cooltown user experience. Technical Report HPL-2001-22, HP Labs, 2001.

[2] Aidan Fuller, Zhong Fan, Charles Day, and Chris Barlow. Digital twin: Enabling technologies, challenges and open research. *IEEE Access*, 8:108952–108971, 2020.

[3] E. H Glaessgen and D. Stargel. The digital twin paradigm for future nasa and us air force vehicles. *AAIA 53rd Structures, Structural Dynamics, and Materials Conference*, 2012.

[4] Michael Grieves. Digital twin: Manufacturing excellence through virtual factory replication. 03 2015.

[5] Michael Grieves. Origins of the digital twin concept, 08 2016.

[6] Dominique Guinard. *A Web of Things Application Architecture: Integrating the Real-World into the Web.* Ph.d. thesis, ETH Zurich, 2011.

[7] Dominique Guinard, Vlad Trifa, Stamatis Karnouskos, Patrik Spiess, and Domnic Savio. Web thing model. W3C Member Submission, 2015. Accessed: 2025-10-17.

[8] Dominique Guinard, Vlad Trifa, Friedemann Mattern, and Erik Wilde. From the internet of things to the web of things: Resource oriented architecture and best practices. In *Architecting the Internet of Things*, pages 97–129. Springer, 2011.

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

[10] Marco Picone, Matteo Martinelli, Samuele Burattini, Andrea Giulianelli, and Alessandro Ricci. The two faces of interoperability: Bridging cyber

and physical spaces with digital twins. In *2025 21st International Conference on Distributed Computing in Smart Systems and the Internet of Things (DCOSS-IoT)*, pages 1–8, 2025.

[11] Alessandro Ricci, Angelo Croatti, Stefano Mariani, Sara Montagna, and Marco Picone. Web of digital twins. *ACM Trans. Internet Technol.*, 22(4), November 2022.

[12] Fei Tao, Meng Zhang, Yushan Liu, and A.Y.C. Nee. Digital twin driven prognostics and health management for complex equipment. *CIRP Annals*, 67(1):169–172, 2018.

[13] Vlad Trifa. *Building Blocks for a Participatory Web of Things: Devices, Infrastructures, and Programming Frameworks*. Ph.d. thesis, ETH Zurich, 2011.

[14] W3C. Launching the web of things interest group. W3C Blog, 2015. Accessed: 2025-10-17.

[15] W3C Web of Things Working Group. Web of things (wot) architecture. W3c recommendation, World Wide Web Consortium (W3C), April 2020. Version 1.0.

[16] W3C Web of Things Working Group. Web of things (wot) thing description. W3c recommendation, World Wide Web Consortium (W3C), April 2020. Version 1.0.