ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

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

STUDIO E PROTOTIPAZIONE DI ECOSISTEMI DI DIGITAL TWIN SU PIATTAFORMA MICROSOFT AZURE DT

Elaborato in
SISTEMI EMBEDDED E INTERNET-OF-THINGS

Relatore
Prof. ALESSANDRO RICCI

Presentata da
NICOLÒ MALUCELLI

Corelatore
Dott. SAMUELE BURATTINI

Anno Accademico 2021 – 2022

alla mia famiglia

Indice

Introduzione

L'avvento dell'Internet of Things, in parallelo con il sempre più diffuso fenomeno della sensorizzazione, ha aperto la strada allo sviluppo di numerose tecnologie, mirate a rendere i semplici oggetti fisici delle entità intelligenti, in grado di condividere attraverso la rete Internet le informazioni catturate.

Uno dei passi logicamente successivi a questo sviluppo tecnologico è costituito dai Digital Twin. L'idea alla base dei Digital Twin è quella di mantenere una copia virtuale di un oggetto fisico in modo da poterne visualizzare le informazioni da remoto senza fastidiosi vincoli spaziali. L'esempio più lampante è probabilmente quello di un razzo spaziale che può essere ininterrottamente monitorato dal centro di controllo in tempo quasi reale.

Mentre inizialmente l'idea di Digital Twin era limitata a quanto appena detto, con il passare del tempo questa visione si è ampliata e l'obiettivo è ora diventato quello di riprodurre in un ambiente virtuale interi sistemi, come ad esempio città, ospedali e così via. Questi sistemi sono costituiti da vere e proprie reti di Digital Twin che sono tra loro connessi attraverso l'uso di relazioni.

A questo proposito le sfide sono presenti e numerose. Prima di tutto, manca è una visione unica e condivisa per quanto riguarda gli ecosistemi di Digital Twin. Una delle proposte in merito a ciò è il Web of Digital Twin, di cui verrà discusso all'interno di questa tesi.

Inoltre, un altro aspetto di cui discutere è la modalità con la quale la componente virtuale viene sincronizzata con quella fisica. Al momento esistono infatti numerose alternative e nessuna prevale sulle altre. Per di più, il concetto di Digital Twin non si riferisce più solamente ai veri e propri oggetti fisici, bensì a qualsiasi risorsa rilevante all'interno di un dato contesto (attività, processi, luoghi, persone, ...). Come è possibile quindi sincronizzare il Digital Twin di risorse che non hanno un proprio sistema embedded associato?

La tesi si apre dunque presentando il concetto di Digital Twin e fornendo le conoscenze di base necessarie per comprendere gli aspetti successivi. A seguito di una breve introduzione che fa il punto sullo stato dell'arte di questa tecnologia, è presentata una delle più utilizzate piattaforme per lo sviluppo di questo tipo di sistemi: Azure Digital Twin.

La tesi prosegue descrivendo quelle che sono le idee alla base del Web of Digital Twin, introducendo i concetti di dinamicità e interconnessione. È inoltre presentata un'architettura orientata agli eventi in grado di soddisfare tutti i requisiti del WoDT, ed è spiegato come tale sistema è in grado di relazionarsi con le applicazioni esterne che vogliono leggere o comandare i Digital Twin che vi appartengono.

In seguito sono confrontati i principi del Web of Digital Twin e le funzionalità offerte da Azure Digital Twin, con l'intento di capire se quest'ultima piattaforma è in grado di soddisfare tutti i requisiti richiesti dal WoDT. In merito a ciò, la tesi si conclude con la prototipazione di un sistema Web of Digital Twin attraverso la piattaforma Azure Digital Twin. Dopo aver definito i requisiti del caso di studio è proposta una possibile soluzione al problema e sono mostrati alcuni esempi di funzionamento.

Essendo la tesi maggiormente incentrata sul concetto di Web of Digital Twin, gli aspetti maggiormente trattati sono le relazioni tra i Digital Twin e la dinamicità del sistema, oltre al fondamentale processo di shadowing. Altre caratteristiche dei Digital Twin come la memorizzazione e la predizione sono invece descritti in maniera meno approfondita.

Capitolo 1

Digital Twin

Prima di cominciare a parlare di Web of Digital Twin è opportuno introdurre il concetto stesso di Digital Twin. Questo primo capitolo comprende un breve riassunto della storia di questa tecnologia, seguito poi da argomenti di carattere più tecnico, come la descrizione della struttura dei Digital Twin e le proprietà che essi devono possedere.

1.1 Origini e storia recente

L'idea di Digital Twin fu presentata per la prima volta nel 2002 da Michael Grieves durante una conferenza riguardante l'industria manifatturiera. Inizialmente il termine utilizzato da Grieves per riferirsi a questa tecnologia era *Mirrored Spaces Model*, tuttavia il concetto alla base era essenzialmente lo stesso. L'idea era quella di creare una copia virtuale di un oggetto, che mantenesse sincronizzato il suo stato durante tutto il ciclo di vita del prodotto: a partire dalla sua creazione, fino alla sua dismessa.

I Digital Twin sono definiti come un insieme di informazioni virtuali che descrivono completamente un oggetto, dal livello microscopico a quello macroscopico, in modo che qualsiasi osservazione che può essere fatta sull'oggetto fisico, possa invece essere fatta su quello virtuale, dando come esito lo stesso risultato [1].

Con il passare del tempo la definizione di Digital Twin si è ampliata, e al giorno d'oggi ad essere rappresentati da Digital Twin non sono più solo i prodotti manifatturieri ma anche luoghi, persone, attività, e più in generale una qualsiasi risorsa che goda di una certa rilevanza all'interno di un dato contesto applicativo. Durante il corso di questa tesi ci riferiremo a queste risorse con il termine *Physical Asset* (PA).

La peculiarità principale dei Digital Twin è l'essere sincronizzati in real-time con l'oggetto fisico a cui si riferiscono. Questa caratteristica rende i Di-

gital Twin appetibili in praticamente qualsiasi contesto applicativo: helthcare [3], industria, smart city [7] e agricoltura [5] sono solo alcune delle possibili applicazioni. In alcuni di questi contesti i Digital Twin trovano già un largo uso, mentre in altri sono ancora in una iniziale fase di sperimentazione.

Possedere una copia digitale di un oggetto, consente di memorizzarne lo stato durante ogni fase del suo ciclo di vita. La grande mole di dati così prodotta, può essere utilizzata da algoritmi di Machine Learning, con l'intento di prevedere quale sarà il comportamento futuro dell'oggetto, a seguito di determinati eventi o condizioni. Questa possibilità, rende la tecnologia dei Digital Twin estremamente appetibile qualora l'oggetto fisico abbia un costo di produzione molto elevato. Non a caso, i Digital Twin hanno trovato un ampio utilizzo in ambito aerospaziale, consentendo di seguire e simulare il comportamento di mezzi e strumenti.

Tuttavia non è necessario pensare a contesti estremi come quello spaziale per trovare un'applicazione pratica ai Digital Twin. In futuro potremmo avere oggetti associati a Digital Twin perfino all'interno delle nostre case, ad esempio per seguire il funzionamento di auto, caldaie o condizionatori. Adottare i Digital Twin può portare grandi vantaggi sia al produttore, che al consumatore. Avere migliaia di prodotti fisici dello stesso tipo che adottano i Digital Twin consente di prevedere se e quando si verificherà un guasto, permettendo di confrontare il comportamento di un dispositivo con quello che avevano i prodotti malfunzionanti. Lato consumatore questo si può tradurre in un migliore servizio di assistenza ed una maggiore affidabilità del prodotto, con conseguenti vantaggi economici per il produttore.

I Digital Twin, inoltre, includono tutti i vantaggi tipici dei prodotti smart, come ad esempio l'aumento delle funzionalità che l'oggetto può offrire. Mentre l'oggetto fisico rimane invariato una volta lasciata la fabbrica, il software ad esso associato rimane in constante evoluzione, rendendo così il prodotto più appetibile rispetto ad altri simili ma non smart.

In generale, un sistema di DT può essere visto come un'infrastruttura che permette alle applicazioni di interagire con gli oggetti fisici attraverso l'uso di interfacce di alto livello, senza che ci si debba preoccupare personalmente dei protocolli di comunicazione o della sincronizzazione a basso livello con il dispositivo.

1.2 Struttura dei Digital Twin

Per definizione, ogni Digital Twin si compone di tre elementi principali (figura 1.1): una componente fisica, una componente virtuale ed un canale di comunicazione che collega le due realtà. Di seguito sono descritte le tre

componente e per ognuna sono individuate le caratteristiche necessarie affinchè il Digital Twin possa funzionare correttamente.

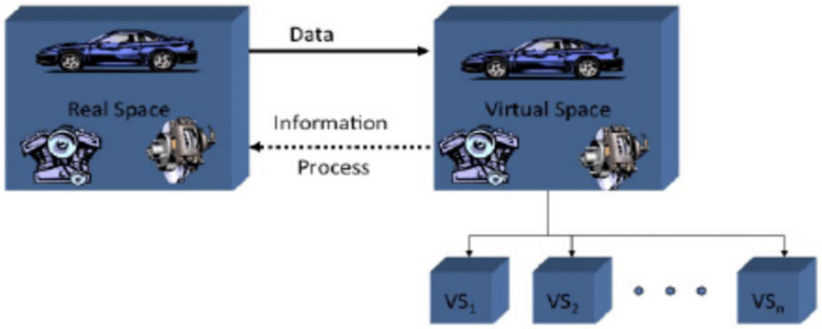


Figura 1.1: Struttura di un Digital Twin come ideata da Michael Grieves

Componente fisica

Come già preannunciato, la componente fisica può essere rappresentata da una qualsiasi risorsa che, presa all'interno del nostro contesto applicativo, risulti utile ai fini della corretta interpretazione del sistema (figura 1.2).

Si supponga ora di voler modellare un sistema di Digital Twin che gestisca una smart city. Poichè quello della città è un contesto applicativo molto vasto, si pensi per ora di modellare la sola gestione del traffico e delle strade. In questo caso ci sono numerosi oggetti fisici di cui può essere interessante tenere traccia: automobili, semafori, controllori di accesso a zone a traffico limitato, e così via. Allo stesso tempo esistono molte altre risorse, non propriamente identificabili in oggetti fisici, di cui però potrebbe comunque essere interessante memorizzarne le informazioni, come strade in cui il traffico è congestionato, incidenti stradali ed aree interessate da lavori in corso, per fare alcuni esempi.

Si noti come in queste due classi ci siano delle caratteristiche comuni ma anche delle differenze:

- ciascuna risorsa, sia che essa appartenga alla prima classe, che alla seconda, può essere interessata a scambiare informazioni con le altre;

- solamente le risorse appartenenti alla prima classe (auto, semafori, controllori di ZTL, ...) sono dotate di un proprio sistema embedded che può produrre dati. Al contrario, le risorse della seconda classe (strade congestionate o cantieri stradali) possono elaborare dati provenienti da altri DT o da applicazioni esterne, ma non possono produrne di nuovi. Questa differenza può essere resa più comprensibile attraverso un esempio. Si

può pensare di creare il Digital Twin di un cantiere stradale attraverso un portale dedicato del comune. Il cantiere stradale potrebbe contenere informazioni riguardanti il luogo in cui si trova, la data di inizio, e quella prevista per la fine dei lavori, e la lista di lavoratori interessati. Non essendo dotato di un proprio sistema embedded, il cantiere stradale non può produrre dati propri, se non attraverso un intervento umano. Al contrario, il Digital Twin di un auto può produrre dati tramite il suo sistema embedded (velocità, numero di passeggeri, ...).

Tipo di risorsa	Risorsa
oggetto fisico	auto semaforo controllore di una ZTL
luogo	cantiere stradale zona residenziale parcheggio
evento	incidente stradale congestione stradale
persona	operaio

Figura 1.2: Esempi di Physical Asset nel contesto della smart mobility

Componente virtuale

La componente virtuale costituisce una vera e propria copia digitale del Physical Asset a cui si riferisce, da qui il termine Digital Twin (DT).

Il DT conserva al suo interno le proprietà del PA di cui ci interessa tenere traccia considerato un dato contesto.

Si prenda ancora una volta in considerazione l'esempio della smart city: il DT di un automobile potrebbe conservare il valore della velocità, la temperatura del motore, se l'auto sta frenando, o quante persone si trovano al suo interno.

Ad un Physical Asset potrebbero essere associati più Digital Twin, ognuno con un proprio livello di astrazione e che mantenga proprietà differenti.

Il Digital Twin permette inoltre alle applicazioni esterne e agli altri Digital Twin, di interagire con l'oggetto fisico che rappresenta, consentendo di leggere informazioni riguardanti lo stato ed eseguire azioni su di esso. Una lampadina potrebbe ad esempio implementare il comportamento di azioni come *turnOn*, *turnOff* o *setIntensity*, accessibili attraverso il proprio Digital Twin.

Come visto prima, uno degli aspetti fondamentali dei Digital Twin è la capacità di fare previsioni riguardo il proprio comportamento futuro. Perchè questo sia possibile è necessario che le informazioni presenti nel Digital Twin siano memorizzate all'interno di un database. Nello specifico, ogni volta che il valore di una proprietà viene aggiornato, nel database deve essere inserito un nuovo record che indichi la proprietà che è cambiata, il suo nuovo valore e l'istante di tempo in cui il cambiamento è avvenuto. Quest'ultimo parametro è fondamentale per permettere di risalire allo stato complessivo della risorsa in ogni momento del passato.

Nel caso si adottasse un'architettura ad eventi per gestire il DT, si può pensare di memorizzare nel database anche gli eventi che hanno influito sul PA e che hanno contribuito a portarlo nello stato in cui si trova.

Comunicazione

La comunicazione tra Physical Asset e Digital Twin avviene in entrambe le direzioni, come mostrato in figura 1.3. A seconda che la comunicazione da Digital Twin a Physical Asset sia automatica o manuale, si parla di Digital Twin o di Digital Shadow [2].

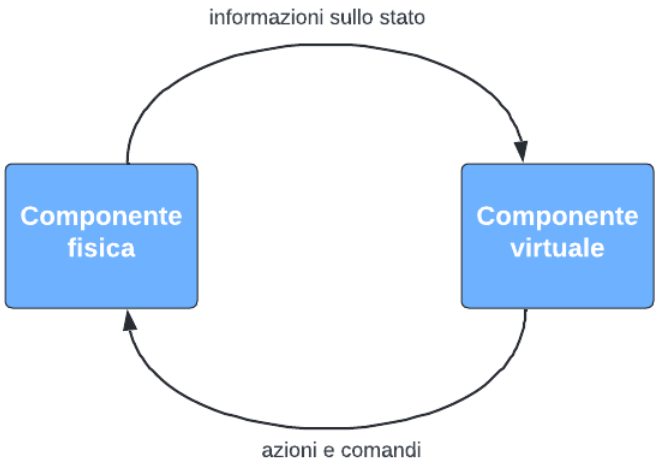


Figura 1.3: Comunicazione tra Digital Twin e Physical Asset

La comunicazione che avviene nel verso PA-DT è fondamentale per la sincronizzazione del Digital Twin con la sua controparte fisica. È importante che la segnalazione del cambiamento di una proprietà (*shadowing*) avvenga il più velocemente possibile, in modo da rendere fedele la rappresentazione offerta dal DT.

Lo scambio di messaggi nel verso DT-PA consente invece di agire sul comportamento del PA attraverso la sua controparte virtuale tramite l'invio di azioni. Questa opzione risulta particolarmente utile quando sul dispositivo fisico sono presenti degli attuatori, come nell'esempio della lampadina descritto poco fa.

Poichè ad uno dei due capi della comunicazione si trova un sistema embedded, la comunicazione avviene prevalentemente attraverso protocolli efficienti che limitano l'utilizzo delle risorse. Rispetto al protocollo HTTP, si preferisce quindi utilizzare il protocollo MQTT, che fornisce prestazioni migliori considerato il contesto in cui ci si trova [8].

1.3 Proprietà fondamentali

Come detto precedentemente, i Digital Twin sono uno strumento che si può rivelare utile in una grande varietà di contesti applicativi. Tuttavia, nonostante questa eterogeneità, possiamo individuare delle proprietà comuni nei Digital Twin che non dipendono dall'ambiente in cui essi sono utilizzati.

L'articolo [4] riconosce le seguenti come le principali:

- *Rappresentatività e Contestualizzazione*: il Digital Twin deve fornire una rappresentazione sufficientemente dettagliata del PA, descrivendone proprietà e comportamento. Questa proprietà è fondamentale considerando che altre proprietà dipendono da essa.

  Preso in considerazione un oggetto, non esiste un solo modo di rappresentarlo. La rappresentazione può variare in base al contesto che stiamo cercando di modellare, ad esempio alcune applicazioni potrebbero richiedere un livello di astrazione minore rispetto ad altre o un altro insieme di proprietà.

- *Riflessione*: è necessario che i cambiamenti che si verificano sul PA siano riflessi anche sulla sua controparte virtuale. L'operazione che permette di soddisfare questa proprietà, prende il nome di *shadowing*.

- *Legame*: Perchè lo *shadowing* possa essere effettuato, è necessario un collegamento tra la componente fisica e la componente virtuale.

  Questa tematica è stata già accennata parlando della struttura dei Digital Twin e sarà nuovamente approfondita con la presentazione del caso di studio.

- *Replicazione*: con replicazione si intende la possibilità di produrre più copie virtuali di uno stesso PA. È imperativo che tutte le copie siano

sincronizzate tra loro, se così non fosse verrebbe a mancare la proprietà fondamentale della riflessione.

- *Persistenza*: il DT deve essere sempre disponibile per interagire con applicazioni esterne e con gli altri DT, anche durante malfunzionamenti del PA e perfino dopo la sua dismessa. La proprietà della persistenza deve quindi essere indipendente dal *legame*.

- *Memorizzazione*: il Digital Twin deve mantenere traccia dello stato del DT durante tutta la sua fase di attività. Ogni volta che il valore di una proprietà cambia o un evento si verifica, il DT deve inserire queste informazioni in un database associandole ad un timestamp. Il timestamp è necessario perchè si possa ricostruire lo stato generale del dispositivo in un dato istante nel passato.

- *Predicibilità*: fare predizioni su un Digital Twin significa utilizzare i dati della storia passata per simulare quale sarà il comportamento dell'oggetto nel futuro al verificarsi di determinati eventi. Perchè questa proprietà possa essere rispettata è necessario che anche la proprietà della memorizzazione lo sia.

- *Componibilità*: un Digital Twin può essere una composizione di altri Digital Twin (relazione di part-of). Attraverso questa dinamica, diventa possibile osservare sia il comportamento del singolo oggetto, sia il comportamento dell'insieme.

- *Servitizzazione*: il Digital Twin è in grado di offrire nuove funzionalità e servizi al semplice oggetto fisico. Queste integrazioni possono rendere il prodotto più appetibile sul mercato rispetto ad altri dello stesso tipo. Vedremo meglio questo concetto quando parleremo di Digital Twin come servizio per il livello applicazione.

- *Accrescimento*: l'accrescimento è una diretta conseguenza della servitizzazione. I prodotti fisici una volta fabbricati non sono più in alcun modo migliorabili, mentre i software invece lo sono. È possibile creare nuove applicazioni che introducono nuove funzionalità o che migliorano quelle esistenti, il tutto senza dover fisicamente intervenire sul prodotto.

Capitolo 2

Una piattaforma per lo sviluppo di DT: Azure Digital Twin

Azure Digital Twin[1] è la soluzione proposta da Microsoft, volta a facilitare lo sviluppo di sistemi di Digital Twin. Essa fornisce un linguaggio che permette la modellazione di Digital Twin e definisce alcune ontologie che agevolano lo sviluppo di sistemi complessi.

Visti i diversi vantaggi che questa piattaforma offre, si è scelto di utilizzarla anche per l'implementazione del caso di studio, come sarà descritto in seguito.

All'interno di questo capitolo saranno analizzate le soluzioni che Azure Digital Twin propone per consentire la progettazione, l'interoperabilità e l'interfacciamento con i Digital Twin, permettendo di rispettare le proprietà dei Digital Twin precedentemente descritte.

2.1 Il Digital Twin Definition Language

Il Digital Twin Definition Language (DTDL) è un linguaggio basato su JSON-LD[2] che permette di definire facilmente modelli di Digital Twin, rendendo allo stesso tempo il codice comprensibile anche da un umano.

Facendo un'analogia con la programmazione ad oggetti, un modello può essere paragonato ad una classe, in quanto permette di descrivere una famiglia di oggetti che condividono proprietà e relazioni comuni. In questa analogia i Digital Twin rappresentano gli oggetti veri e propri in quanto istanze dei modelli.

Tra le altre cose, DTDL supporta inoltre l'ereditarietà, questo significa che è possibile creare nuovi modelli che estendono modelli già esistenti e ne specia-

[1] https://azure.microsoft.com/en-us/services/digital-twins/

[2] https://json-ld.org/

lizzano il comportamento. Restando in tema mobilità smart, si può pensare di modellare l'insieme di tutti i veicoli attraverso il modello "veicolo" e creare poi i modelli "automobile" e "autobus" che lo specializzano focalizzandosi su aspetti più dettagliati, come il proprietario o il conducente nel caso dell'automobile o il numero di passeggeri e la tratta nel caso dell'autobus. Dato che DTDL supporta l'ereditarietà multipla, è anche possibile che un modello estenda il comportamento di due o più modelli.

2.1.1 Elementi di un modello

L'elemento di base di ogni modello è l'interfaccia. L'`interface` può essere paragonata alle classi della programmazione ad oggetti. Le interfacce sono identificate da un `id` che deve essere univoco all'interno di un'istanza di Azure Digital Twin, e possono indicare anche un "friendly name".

Ogni interfaccia definisce un modello, e ne descrive la struttura attraverso una serie di proprietà, telemetrie, relazioni e componenti, che insieme costituiscono il contenuto (`contents`) del modello.

Come detto poco fa, un'interfaccia può estendere un insieme di altre interfacce attraverso la keyword `extends`, seguita dalla lista delle interfacce che si desidera estendere. Un'interfaccia può solo aggiungere informazioni alle interfacce che estende, non può rimuovere proprietà o relazioni, né tantomeno ridefinirne i concetti.

Nel corso di questa sezione sarà utilizzata l'interfaccia *Vehicle* (listato 2.1) per dare forma ai concetti visti, e verranno aggiunti i vari elementi mano a mano che saranno spiegati.

```
{
    "@id": "dtmi:com:example:vehicle;1",
    "@type": "Interface",
    "displayName": "Vehicle",
    "contents": [

    ],
    "@context": "dtmi:dtdl:context;2"
}
```

Listato 2.1: Schema dell'interfaccia `Vehicle`

Di seguito sono presentati gli elementi che possono essere utilizzati per definire un modello, ovvero: proprietà, relazioni, telemetrie e componenti.

Property

Volendo continuare l'analogia con la programmazione ad oggetti, le proprietà rappresentano i campi di una classe e sono costituiti da un nome e da un tipo. Il tipo, nel DTDL prende il nome di `schema`, in quanto la keyword `type` in JSON è già utilizzata per definire la struttura del modello. Il campo `schema` può essere sia un tipo primitivo, come `integer`, `boolean` o `double`, ma anche un tipo complesso, per esempio `Enum` o `Object`.

DTDL introduce inoltre il concetto di unità di misura per un insieme di tipi semantici noti, come temperatura, luminosità e lunghezza, per citarne alcuni. Ad esempio la temperatura può essere rappresentata in gradi Celsius, Fahrenheit e Kelvin, e la lunghezza in metri, centimetri, pollici e così via.

Nell'esempio seguente sono descritte alcune proprietà che l'interfaccia *Vehicle* precedentemente presentata (listato 2.1) potrebbe contenere.

```
{
    "@type": ["Property", "Velocity"],
    "name": "speed",
    "unit": "kilometrePerSecond",
    "schema": "float"
},
{
    "@type": "Property",
    "name": "nPassengers",
    "schema": "integer"
}
```

Listato 2.2: Esempio delle proprietà di un modello

Telemetry

Come le proprietà, anche le telemetrie hanno i campi `name` e `schema` e possono avere un tipo semantico e un'unità.

Ciò che differenzia le telemetrie dalle proprietà è che le prime sono più simili ad un flusso di dati: se il valore non viene consumato in tempo esso andrà perso e non potrà più essere recuperato. Utilizzando le telemetrie non è ad esempio possibile eseguire query sullo stato corrente del dispositivo. Per questo motivo, all'interno dei modelli sono maggiormente utilizzate le proprietà, in quanto consentono di tenere traccia dell'andamento del valore nel corso del tempo, mentre le telemetrie sono maggiormente utilizzate per gestire l'input e l'output dei dati, ad esempio attraverso il pattern observer.

Il termine telemetria non è esclusivo del DTDL, al contrario è abbastanza comune incontrarlo all'interno dell'ecosistema Azure in diversi altri contesti.

Le telemetrie sono ampiamente utilizzate per scatenare eventi, come l'aggiornamento di un Digital Twin, e notificare tali cambiamenti alle applicazioni esterne.

Relationship

Le relazioni consentono di creare legami tra Digital Twin e sono identificate da un nome che esprime il concetto della relazione. Ad esempio un veicolo può avere l'associazione *owner* che punta al Digital Twin della persona che possiede il mezzo.

Le relazioni possono specificare o meno un `target`, ovvero l'id del modello a cui puntano. Se il campo `target` dovesse mancare, si parlerebbe di relazione *non-targeted* e qualsiasi Digital Twin potrebbe essere la destinazione del legame.

Così come le interfacce, anche le relazioni possono avere delle proprietà che ne descrivono alcuni aspetti. Riprendendo il caso di prima, potrebbe essere interessante conoscere da quando una persona è proprietaria dell'auto (listato 2.3).

```
{
  "@type": "Relationship",
  "name": "owner",
  "target": "dtmi:com:adt:dtsample:person;1",
  "properties": [
    {
      "@type": "Property",
      "name": "since",
      "schema": "dateTime"
    }
  ]
}
```

Listato 2.3: Relazione *targeted* con proprietà per l'interfaccia *Vehicle* precedente

Data un'istanza di Azure Digital Twin è possibile rappresentare il grafo che mostra i legami tra i Digital Twin. In un grafo di questo tipo, i DT rappresentano i nodi mentre le relazioni sono gli archi che collegano i nodi tra loro. Anche le relazioni, così come le proprietà, possono evolvere nel tempo a seconda del sistema, di conseguenza anche il grafo dei Digital Twin cambierà dinamicamente.

Component

I `Component` permettono di annidare un'interfaccia all'interno di un'altra, descrivendo quindi un legame di part-of. Utilizziamo i `Component` quando l'esistenza del componente stesso dipende direttamente dall'esistenza dell'interfaccia che li racchiude e non avrebbe quindi senso considerarli separatamente.

Il massimo livello di annidamento dei `Component` è uno, ciò significa che non è possibile definire una componente all'interno di un'altra componente, questo per evitare cicli.

Come per le proprietà e le telemetrie, anche per i componenti si chiede di specificare un nome ed uno schema. Questa volta però, a differenza di prima, lo schema non è un tipo primitivo o complesso, ma è l'id di un'altra interfaccia.

Nonostante relazioni e componenti possano sembrare simili, differiscono tra loro per diversi aspetti:

- mentre i componenti sono salvati per valore, le relazioni contengono unicamente un riferimento a un'interfaccia già esistente;
- le relazioni possono definire loro proprietà mentre i componenti si limitano a fornire le proprietà dell'interfaccia indicata dal campo `schema`;
- i componenti hanno sempre un target (`schema`), mentre le relazioni possono anche non indicarlo (*non-targeted*);
- i componenti sono più simili alle proprietà di quanto non lo siano alle relazioni. Essi consentono di modellare unicamente legami di tipo part-of mentre le relazioni permettono di descrivere qualsiasi tipo di legame.

Nell'esempio che segue è descritto il componente *engine* per l'interfaccia *vehicle* mostrata precedentemente.

```
{
    "@type": "Component",
    "name": "engine",
    "schema": "dtmi:com:adt:dtsample:engine;1"
}
```

Listato 2.4: Esempio di componente per l'interfaccia *vehicle* precedente

2.1.2 Ontologie nella progettazione di Digital Twin

Nell'informatica un'ontologia è una rappresentazione condivisa e accettata di un certo dominio applicativo. Nello specifico, in Azure Digital Twin, le

ontologie sono insiemi di modelli utilizzati per descrivere domini applicativi comuni, come le smart city o gli smart building.

Adottare le ontologie permette di risparmiare sul processo di progettazione, riducendo tempi e costi, utilizzando al tempo stesso modelli completi ideati da esperti del settore.

Le ontologie offrono una visione ampia di un dato dominio applicativo, tuttavia a seconda di quello che è lo scopo del sistema, potrebbe essere necessario focalizzarsi su aspetti differenti che potrebbero non essere descritti dettagliatamente dall'ontologia. Nulla vieta quindi di estendere le ontologie attraverso gli elementi visti in precedenza, creando nuovi modelli che estendono quelli già presenti o aggiungendo proprietà e relazioni ai modelli originali, in modo da ottenerne di più adatti alle nostre specifiche.

2.2 Sincronizzazione con i dispositivi fisici

La sincronizzazione, o *shadowing*, è la procedura che consente al Digital Twin di essere sempre aggiornato con lo stato del dispositivo fisico. Volendo fare un richiamo alle proprietà dei Digital Twin descritte in precedenza, il processo di sincronizzazione corrisponde alla proprietà di *riflessione* e necessita di un *legame* tra la componente fisica e la componente digitale.

Azure Digital Twin offre diverse modalità per quanto riguarda la sincronizzazione tra componente fisica e componente virtuale. Ciò che accomuna le varie alternative è che ognuna fa uso delle REST API per l'aggiornamento vero e proprio delle proprietà dei Digital Twin. All'interno di questa sezione saranno mostrate due di queste strategie: la prima fa uso di un'altra piattaforma dell'ecosistema Azure, ovvero IoT Hub[3], mentre la seconda no.

2.2.1 Sincronizzazione attraverso IoT Hub

IoT Hub è una piattaforma che fa da tramite tra i sistemi embedded e le applicazioni che vogliono interagire con esse, facilitando lo scambio di messaggi ed introducendo affidabilità e sicurezza alle comunicazioni.

Quando si crea un nuovo dispositivo all'interno di un'istanza di IoT Hub è possibile optare per due livelli di sicurezza: il primo basato sui SAS token ed il secondo sulla certificazione X.509.

Il SAS token può essere generato attraverso l'interfaccia a linea di comando di Azure CLI[4], indicando l'id del dispositivo e il nome dell'hub in cui esso si trova:

[3] https://azure.microsoft.com/en-us/services/iot-hub/

[4] https://docs.microsoft.com/en-us/cli/azure/what-is-azure-cli

```
az iot hub generate-sas-token --device-id XXX --hub-name XXX
```

Listato 2.5: Generazione del SAS token

Una volta configurato il token, è possibile inviare le telemetrie all'IoT Hub. Per la comunicazione tra dispositivo e IoT Hub è possibile usare uno dei seguenti protocolli: MQTT, AMQP e HTTPS. Ogni messaggio che inviamo all'IoT Hub, per essere autenticato, deve contenere nell'header il campo *Authentication*, il cui valore è il SAS token precedentemente generato. Per rendere più sicura la comunicazione, il token ha una validità limitata in termini di tempo, successivamente sarà necessario generarne uno nuovo.

Una volta stabilita una connessione sicura tra il dispositivo fisico e l'IoT Hub e ricevute le telemetrie, è necessario aggiornare l'istanza del Digital Twin corrispondente. Per farlo si può utilizzare una Azure Function che prende in input le telemetrie dall'IoT Hub e tramite l'uso delle API aggiorna i valori delle proprietà dei Digital Twin veri e propri.

2.2.2 Una possibile strategia alternativa

Come si può notare, IoT Hub fa da tramite tra il dispositivo fisico vero e proprio e la sua controparte digitale, tuttavia è possibile ottenere lo stesso risultato senza utilizzare il servizio di Azure, anche se a quel punto sarà necessario occuparsi personalmente di garantire la sicurezza e l'autenticazione delle comunicazioni all'interno del sistema.

Una delle alternative più semplici consiste nell'unire in una singola applicazione le funzioni che prima erano svolte dall'IoT Hub e dalla Azure Function, rispettivamente:

- la comunicazione con il dispositivo
- l'aggiornamento del Digital Twin

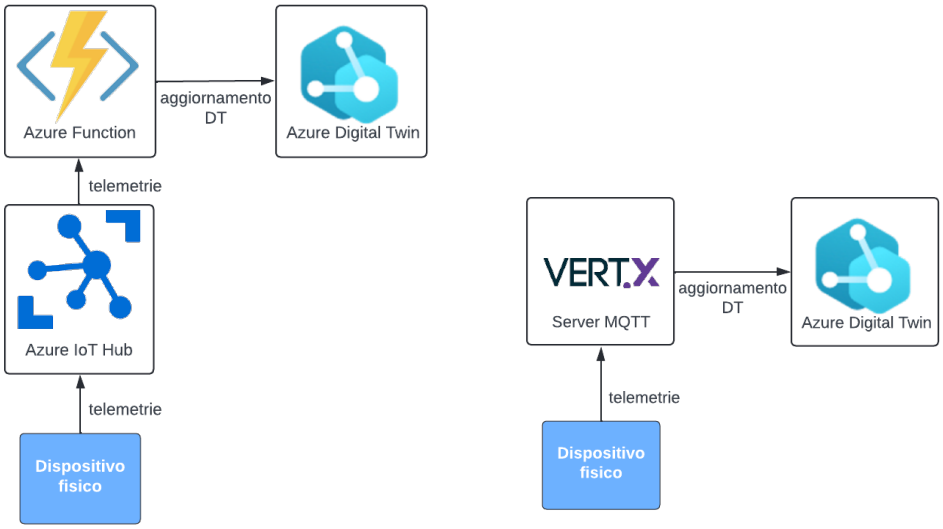
Azure Digital Twin offre un insieme di API accessibili attraverso SDK disponibili per una grande varietà di linguaggi di programmazione, tra i quali Java, C# e Python. Per costruire il server si può ad esempio utilizzare Java in combinazione con il toolkit *Vert.x*[5].

Nell'immagine che segue (figura 2.1) sono mostrate le due soluzioni proposte: a sinistra l'architettura basata unicamente su servizi dell'ecosistema Azure, mentre a destra la soluzione che utilizza un semplice server.

In particolare, questa seconda opzione (figura 2.1b), è stata preferita alla prima per l'implementazione del caso di studio.

[5]https://vertx.io/

Un altro aspetto fondamentale di cui tenere a mente è la scalabilità del sistema. Se il sistema conta solo pochi dispositivi fisici, che inviano e ricevono dati da un server, allora una soluzione equivale l'altra in termini di efficienza. Tuttavia se il sistema dovesse contare centinaia, o anche migliaia di dispositivi, una semplice applicazione difficilmente potrebbe soddisfare le richieste di tutti. In questo contesto IoT Hub si rivela ancora una volta, un'ottima soluzione, in quanto consente una corretta scalabilità del sistema senza che il progettista si debba occupare personalmente di essa.



(a) Soluzione con IoT Hub (b) Soluzione senza IoT Hub

Figura 2.1: Soluzioni per la sincronizzazione a confronto

2.3 Gestione dello storico

Come già detto, uno degli aspetti principali dei Digital Twin è la possibilità di prevedere il comportamento futuro dell'oggetto sulla base di come esso ha reagito agli eventi passati. Perchè questo sia possibile è necessario memorizzare lo stato assunto dal Digital Twin durante tutto il suo corso di vita.

Conservare i dati riguardanti oggetti fisici può inoltre avere secondi fini: è ad esempio possibile effettuare analisi statistiche sui dati, per stabilire strategie mirate ad aumentare l'efficienza dell'ambiente in cui i DT operano.

Azure Digital Twin non si occupa personalmente di mantenere lo storico dei dati, ma si affida ad un'altra piattaforma dell'ecosistema Azure: Azure Data Explorer[6].

[6]https://azure.microsoft.com/en-us/services/data-explorer/

Quando il Digital Twin riceve una telemetria, nelle modalità che abbiamo visto prima, aggiorna le proprietà indicate e genera una nuova telemetria contenente il proprio stato aggiornato.

In ascolto delle telemetrie provenienti dall'istanza di Azure Digital Twin, è presente un Event Hub. L'Event Hub è un'altra piattaforma dell'ecosistema Azure che consente di ricevere ed inoltrare eventi anche ad un alto livello di scalabilità. L'Event Hub inoltra quindi il messaggio ad un'istanza del Data Explorer che memorizza il nuovo valore delle proprietà, specificando l'id del DT a cui corrispondo ed un timestamp che indica quando il DT ha ricevuto la prima telemetria.

Normalmente Azure Digital Twin permette di effettuare interrogazioni solamente sullo stato corrente del sistema, in quanto, come detto, non si occupa della memorizzazione dello storico. Se si decide di utilizzare la feature della Data History vista sopra, diventa però possibile svolgere query anche sullo stato passato del sistema, grazie ad un plugin di Azure Digital Twin per il Data Explorer.

Il linguaggio di interrogazione è un linguaggio SQL-like e le query possono essere effettuate sia tramite la piattaforma stessa di Azure Digital Twin, sia attraverso l'uso delle API.

Quando il valore di una proprietà cambia, nel cluster è aggiunto un solo record corrispondente alla proprietà che è stata aggiornata. Pertanto, prendendo un istante t, per ricostruire qual era lo stato del DT in quel determinato momento è necessario selezionare per ogni proprietà, l'ultimo record con timestamp minore a t.

2.4 Interagire con i Digital Twin

Azure Digital Twin fornisce due diverse strategie per interagire con i Digital Twin del sistema. La prima consiste nell'utilizzare delle REST API, mentre la seconda nell'utilizzare l'interfaccia di Azure Digital Twin Explorer. Tra queste la prima soluzione risulta essere la più utilizzata, in quanto consente ad applicazioni esterne di interfacciarsi con il sistema.

Azure DT Explorer può invece essere utilizzato, per ottenere un riscontro grafico sullo stato corrente del sistema.

Durante l'implementazione del caso di studio sono state utilizzate entrambe le alternative.

All'interno di questa sezione sono introdotte le due appena citate modalità di interazione ed è descritto il principale elemento alla base della comunicazione tra l'istanza di Azure Digital Twin e le componenti esterne, ovvero la notifica.

2.4.1 Interazione tramite REST API

Nelle pagine precedenti, è stato descritto come Azure Digital Twin permette di sincronizzare i Digital Twin con le loro controparti fisiche e come è possibile memorizzare lo storico del sistema attraverso la feature della Data History per eseguire interrogazioni su di esso. Perchè tutto ciò sia possibile è necessaria una tecnologia che permette alle applicazioni esterne di interagire con i Digital Twin. Questa tecnologia sono le REST API.

Per facilitare l'uso delle API, Azure Digital Twin mette a disposizione degli SDK per diversi linguaggi di programmazione, come: Java, JavaScript, C#, Python e Go, ma in alternativa è possibile utilizzare le API direttamente attraverso richieste HTTP.

Uno degli aspetti chiave per un sistema di Digital Twin è la sua dinamicità: spesso siamo interessati a creare a run-time nuovi Digital Twin o a distruggerne di esistenti. Riprendendo l'esempio di prima della smart city, possiamo immaginare che il DT di un veicolo venga generato durante la sua immatricolazione e sia distrutto con la sua demolizione.

Un'altra operazione comune è quella di modificare i target delle associazioni, ad esempio se un'auto viene venduta la relazione che la lega al proprietario deve essere aggiornata, facendola puntare al DT del nuovo proprietario.

Le API consentono anche di eliminare, aggiornare e aggiungere nuovi modelli, questo può essere utile nel caso in si vogliano aumentare le funzionalità offerte dall'applicazione, o migliorare quelle già esistenti aggiungendo ulteriori feature.

2.4.2 Azure Digital Twin Explorer

Azure offre anche un'alternativa grafica per interagire con i Digital Twin: la piattaforma Azure Digital Twin Explorer. Attraverso questa piattaforma è possibile importare nuovi modelli e istanziare nuovi Digital Twin, nonchè aggiornare manualmente il valore delle proprietà e i target delle relazione, ed eseguire query sullo stato corrente del sistema.

Come detto poco fa, una delle caratteristiche principali dei Digital Twin è la loro dinamicità, pertanto poter agire manualmente sui DT può sembrare poco utile, tuttavia non è così. Questo strumento può infatti essere usato per testare il sistema e simularne il comportamento, inoltre potrebbe essere utile creare a mano i primi Digital Twin che costituiscono la struttura di base del nostro sistema, attorno alla quale creare poi i nuovi DT a run-time.

Un'altra funzionalità dell'explorer di Azure Digital Twin è visualizzare in tempo reale il grafo che collega i Digital Twin. In questo grafo i DT costituiscono i nodi mentre le relazioni sono rappresentate dagli archi. Aggiornando

il target di una relazione, la modifica si riflette anche sul grafo, pertanto si può utilizzare questo strumento per tenere traccia dello stato corrente del sistema. Eseguendo le query si ottiene un riscontro grafico, in quanto dal grafo originale rimangono visibili soltanto i Digital Twin che soddisfano le condizioni dell'interrogazione.

2.4.3 Il sistema di notifiche

Azure Digital Twin adotta un modello basato sulle notifiche per la comunicazione tra due DT e tra i DT e le applicazioni esterne. Ogni notifica è generata da un evento e conterrà informazioni differenti a seconda del tipo di evento che l'ha generata.

In Azure Digital Twin distinguiamo quattro tipi di notifiche:

- **Aggiornamento di un DT:** questa notifica è generata quando il valore di una proprietà di un Digital Twin cambia, ma anche quando ad essere aggiornato è il modello.
- **Aggiornamento di una relazione:** è generata quando il target di una relazione cambia o viene rimosso, o quando il valore di una proprietà interna alla relazione viene aggiornato.
- **La ricezione di una telemetria:** abbiamo già accennato all'uso delle telemetrie parlando della sincronizzazione dei DT. Questo tipo di notifica può essere usata per propagare la telemetria all'esterno o ad un altro DT.
- **La creazione o la distruzione di un DT:** i DT generano una notifica negli istanti della loro creazione e della loro distruzione. Questo può essere utile per comunicare alle applicazioni esterne l'esistenza di un nuovo DT o notificare loro che un DT non esiste più.

In precedenza è stato descritto un esempio di comunicazione tra DT e applicazioni esterne, nello specifico, parlando della gestione dello storico. È stato detto che quando il valore di una proprietà viene aggiornato, il suo cambiamento è notificato ad un Event Hub attraverso una notifica. L'Event Hub svolge il ruolo di endpoint per la comunicazione, ricevendo messaggi da più sorgenti ed inoltrandoli ai destinatari. Nel caso della feature della Data History non sono necessari altri intermediari: l'istanza del Data Explorer è direttamente connessa all'Event Hub ed aggiorna così il cluster inserendo i nuovi dati.

Lo scambio di messaggi tra Digital Twin avviene in maniera simile: è possibile creare una Azure Function che reagisca ai messaggi inviati dall'Event Hub eseguendo azioni sui Digital Twin attraverso l'uso delle API viste precedentemente. Questa funzionalità è utile in quanto permette di reagire a ciò che

accade all'interno dell'ambiente di Digital Twin. Ad esempio potrebbe essere utile creare un nuovo DT o modificare una relazione al seguito del verificarsi di determinate condizioni.

Senza un'architettura di questo tipo, sarebbe necessario interrogare periodicamente il sistema in modo da verificare eventuali cambiamenti ad esempio nei valori delle proprietà. Utilizzare un'architettura ad eventi permette quindi di aumentare l'efficienza del sistema senza dovere rinunciare alla sua complessità.

Capitolo 3

Un ecosistema di DT: Web of Digital Twin

Allo stato attuale non esiste una soluzione unica e condivisa per la progettazione dei Digital Twin, al contrario esistono molte piattaforme che offrono funzionalità più o meno avanzate per lo sviluppo di questo tipo di sistemi. Una di queste piattaforme è proprio Azure Digital Twin, già introdotta nelle pagine precedenti, ma le aziende attive in questo ambito sono numerose, così come le soluzioni proposte.

La maggior parte delle piattaforme per lo sviluppo di Digital Twin adotta un sistema chiuso. Nel corso di questo capitolo verrà analizzato come il Web of Digital Twin [6] si propone come soluzione a questo e ad altri problemi, definendo un proprio modello ed una propria architettura basata sugli eventi. Uno degli aspetti al centro del Web of Digital Twin è proprio l'idea di realizzare un sistema aperto e distribuito, caratterizzato dalla dinamicità e dall'interconnessione delle parti che lo costituiscono.

3.1 Un sistema dinamico e interconnesso

Nella visione del WoDT, ogni Digital Twin è rappresentato da un'istanza di un modello. Il modello definisce la struttura del DT, ovvero indica attraverso quali proprietà e quali relazioni, il Physical Asset è descritto all'interno dell'ambiente virtuale. Come detto in precedenza, un PA può essere descritto da più di un modello: ogni modello fornisce una diversa vista sulla risorsa, ovvero descrive il PA attraverso diverse proprietà e relazioni, a seconda del contesto in cui esso è utilizzato.

Il concetto di modello è già stato incontrato parlando di Azure Digital Twin e di DTDL. WoDT e Azure Digital Twin hanno infatti diverse caratteristiche comuni, che saranno analizzate più avanti in una sezione dedicata.

Ogni Digital Twin all'interno del WoDT è descritto attraverso un Knowledge Graph, che può essere rappresentato mediante gli strumenti del Web semantico, come ad esempio RDF. Il Knowledge Graph permette di visualizzare attraverso un grafo lo stato di un Digital Twin, ovvero il valore delle sue proprietà e le relazioni con gli altri DT appartenenti al WoDT.

Parlando di sistema dinamico si intende dire che la rappresentazione virtuale si evolve di pari passo con il mondo reale, mediante un processo chiamato *shadowing*. Lo shadowing consiste nel tenere sincronizzato lo stato del Digital Twin con quello del rispettivo Physical Asset. Perchè lo shadowing sia possibile, è necessario che il PA catturi gli eventi e li inoltri alla propria controparte virtuale, che si occupa quindi di aggiornare lo stato sulla base dello stato corrente e dell'evento ricevuto. A seconda dell'evento registrato e dello stato in cui il DT si trova, il DT evolverà di conseguenza.

La dinamicità ovviamente non si limita al solo aggiornamento dei valori delle singole proprietà: a poter cambiare sono infatti anche i riferimenti delle relazioni. Ciò significa che le connessioni tra i Digital Twin possono cambiare a seconda di ciò che accade nel mondo fisico. Nel caso della smart mobility si può pensare all'ipotesi in cui un'auto può cambiare proprietario. Questo cambiamento deve essere di conseguenza riflesso anche nella rappresentazione virtuale, aggiornando la destinazione del collegamento in modo che punti al DT del nuovo proprietario. Se così non fosse, la rappresentazione reale e quella virtuale non coinciderebbero, vanificando il concetto stesso di Digital Twin.

Aggiornare proprietà e relazioni si riflette di conseguenza anche sul Knowledge Graph. Il KG si evolve infatti parallelamente al mondo reale, continuando a fornire una visione aggiornata di ciò sta accadendo. Come detto anche in precedenza, parlando delle caratteristiche che deve avere un Digital Twin, uno degli aspetti fondamentali risulta essere la memorizzazione dello storico. Conservare le informazioni passate del sistema, quindi relazioni, proprietà ed eventi, consente di ricostruire lo stato del sistema ad ogni istante di tempo, permettendo in caso di problemi o malfunzionalmenti, di analizzare le cause che hanno portato il sistema nello stato in cui si trova. Graficamente questo si traduce nella possibilità di navigare il Knowledge Graph a partire da quando il sistema è stato reso attivo, fino all'istante di tempo corrente.

3.1.1 Creazione dei Digital Twin

I singoli Digital Twin possono evolvere sia in termini di proprietà che di relazioni con altri DT, ma la dinamicità del WoDT non si limita a questo. Perchè il WoDT possa offrire una rappresentazione fedele della realtà, è necessario poter creare dinamicamente, ovvero a run-time, nuovi Digital Twin, e allo stesso tempo eliminare DT già esistenti.

Riprendendo come esempio il caso della smart mobility, si può pensare di creare il Digital Twin di un'auto al momento dell'immatricolazione ed eliminarlo al momento della demolizione. Se non ci fosse la possibilità di creare o eliminare Digital Twin a run-time, le nuove auto prodotte dopo l'avvio del sistema non potrebbero avere una propria rappresentazione digitale. Allo stesso modo, se non ci fosse la possibilità di eliminarli, il sistema potrebbe pensare che un'auto demolita in realtà esista ancora.

Distinguiamo la creazione dinamica dei Digital Twin in due modalità: tramite shadowing e tramite applicazione.

Creare un DT tramite shadowing significa creare un nuovo DT a partire da uno già esistente, consentendo di fornire una vista più semplificata a seconda di ciò che ci serve fare con il nuovo DT. Questa strategia può essere utilizzata per ridurre il carico di lavoro sul Physical Asset (figura 3.1a), infatti si può pensare di creare un DT che sia direttamente connesso al PA e che raccolga tutte le informazioni utili per ogni contesto, per poi creare tramite shadowing altri Digital Twin, ognuno dei quali catturi un aspetto diverso a seconda delle necessità. Così facendo il PA comunica i dati ad un solo DT, permettendo di risparmiare in termini di risorse energetiche. Il DT "padre" non avendo bassi limiti di risorse, si può occupare agevolmente di sincronizzare i DT più specializzati. L'alternativa a questa strategia è far comunicare tutti i DT con il PA (figura 3.1b), ma in questo caso si ha uno spreco inutile di risorse in quanto potremmo dover comunicare gli stessi eventi a più DT creando ridondanza.

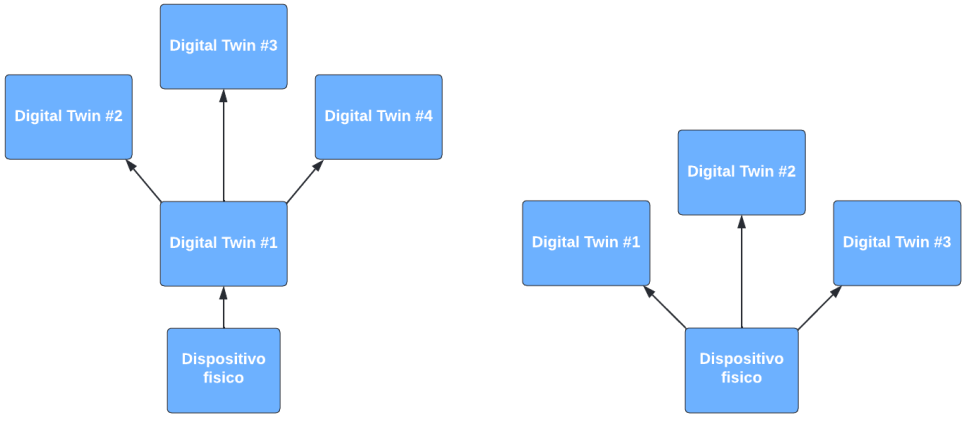


Figura 3.1: due diverse strategie di shadowing

Nel caso della creazione di una nuova auto non è però questa l'operazione necessaria. In questo caso ciò che risulta più comodo è la possibilità di creare dinamicamente nuovi DT attraverso un software che si interfacci con il WoDT.

Si può pensare di disporre la creazione del nuovo DT al momento dell'immatricolazionde dell'auto: quando un'auto viene immatricolata è creata anche la sua istanza virtuale, e il DT del proprietario è associato ad essa tramite un'opportuna relazione.

Seppure esista la possibilità di creare nuovi Digital Twin dinamicamente, questa non è l'unica soluzione. Rimane infatti possibile creare i Digital Twin in maniera statica, ovvero prima che il sistema avvii la propria esecuzione. Questa alternativa è spesso utilizzata per creare i Digital Twin che costituiscono l'ossatura del sistema, in modo da poter creare poi dinamicamente nuovi DT appoggiandosi ad una struttura già esistente. Nell'esempio della smart mobility, qualora fosse necessario, si può pensare di creare staticamente i Digital Twin dei quartieri che costituiscono la città. Successivamente, ai quartieri si potranno legare dinamicamente i semafori attraverso una relazione di appartenenza.

3.2 L'architettura del WoDT

All'interno di questa sezione sarà analizzata una delle possibili architetture che il WoDT potrebbe adottare. Nello specifico si parla dell'architettura di tipo event-driven proposta all'interno dell'articolo "Web of Digital Twin" [6] e rappresentata in figura 3.2. Questa soluzione ovviamente non è l'unica possibile, in quanto una qualsiasi proposta che riesca a soddisfare tutti i requisiti dei Digital Twin e del WoDT può ritenersi valida.

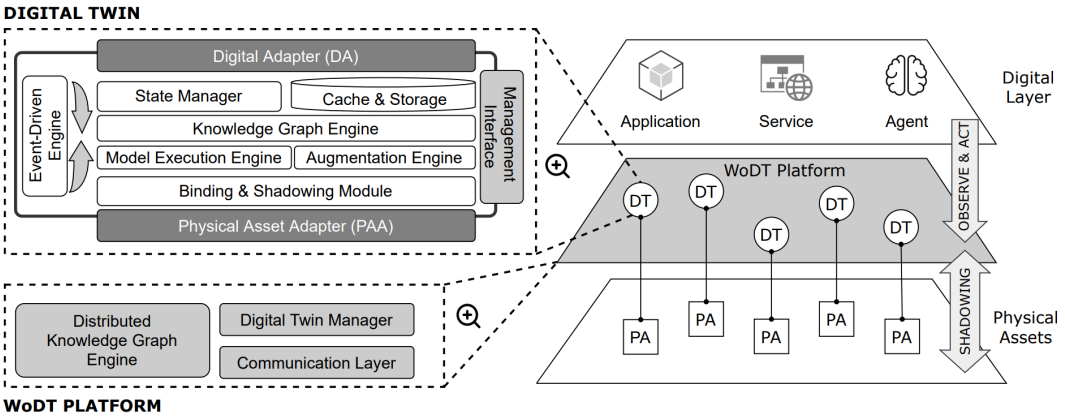


Figura 3.2: Architettura proposta per il WoDT [6]

Come sarà descritto in seguito, il WoDT agisce da strato intermedio tra i Physical Asset ed il livello applicazione, permettendo una comunicazione

bidirezionale con entrambi. In questa analisi sarà distinta l'architettura del DT vero e proprio da quella del WoDT che li ingloba.

3.2.1 Architettura di un Digital Twin

Come si può notare dalla figura 3.2, sono presenti tre componenti che permettono al Digital Twin di comunicare con l'esterno. Queste tre componenti sono: il Physical Asset Adapter, il Digital Adapter e la Management Interface.

il Physical Asset Adapter (PAA) si occupa della comunicazione con il Physical Asset. Nello specifico, esso cattura gli eventi provenienti dal PA, chiamati e_{PA}, e li traduce in eventi interni e_I. La conversione da e_{PA} a e_I è uno dei requisiti del WoDT ed è necessaria in quanto non esiste un'unica tipologia di PA e di conseguenza gli eventi generati dal sistema embedded possono essere di diverso tipo e formato. Diversi PA potrebbero infatti adottare diverse architetture e diversi protocolli a seconda della loro complessità o del loro contesto applicativo.

Il Digital Adapter (DA) ha una funzione simile al PAA. Mentre il PAA si occupa di gestire le comunicazioni con il PA, il DA ha il compito di gestire le comunicazioni con le applicazioni esterne. Come il PAA, anche il DA si occupa di convertire gli eventi da un formato di rappresentazione a un altro, nello specifico converte gli eventi e_I in eventi del Digital Twin, e_{DT}. Le applicazioni esterne possono richiedere al DT di eseguire una o più operazioni ed il loro risultato sarà comunicato al PA attraverso il DA. L'insieme di operazioni che può essere eseguito su un Digital Twin è definito dalla Management Interface, la quale gestisce anche le interrogazioni riguardanti il ciclo di vita del DT.

Come detto, un'applicazione può anche eseguire azioni su un Digital Twin, e quindi sul relativo PA: per farlo comunica le funzioni sotto forma di evento e_{DT} al DA, il quale si occupa di convertirle in eventi interni e_I che a loro volta saranno riconvertiti dal PAA in eventi e_{PA} e comunicati finalmente al PA.

Prima di spiegare le prossime componenti è necessario introdurre il funzionamento dell'Event-driven Engine (EE). Questa componente consente agli altri moduli del DT di comunicare tra loro attraverso gli eventi interni e_I. Poichè questa soluzione è basata su un modello event-driven, anche la strategia di comunicazione è di tipo event-driven. Qualora si decidesse di utilizzare un altro tipo di architettura, questa componente andrebbe rivalutata, così come la comunicazione tra i moduli interni.

Ora che abbiamo stabilito come le componenti del DT comunicano tra loro, possiamo procedere con l'analisi dei moduli interni al DT.

il Binding & Shadowing Module (BSM) si occupa di mantenere sincronizzato il DT con il relativo PA e di gestire l'intero ciclo di vita del Digital Twin. Perchè l'operazione di shadowing sia possibile, è necessario che l'EE comuni-

chi al BSM gli eventi catturati dal PA attraverso il PAA sottoforma di eventi interni e_I.

L'elemento centrale nell'architettura del DT è sicuramente il Model Execution Engine (MEE). Questo modulo si occupa di gestire il funzionamento di tutte le altre componenti, definendo ad esempio quali eventi catturare e come tali eventi debbano influire sul funzionamento del Digital Twin.

Una componente che lavora a stretto contatto con l'MEE è lo State Manager (SM), il quale si occupa di gestire gli aggiornamenti di stato del DT. Il nuovo stato è calcolato sulla base del modello, valutando lo stato corrente del DT e l'evento e_{PA} che è stato ricevuto.

Una volta calcolato il nuovo stato, esso viene salvato dal modulo di Cache & Storage (CS). Questo modulo consente di rispettare il principio della memorizzazione dei DT, e di conseguenza permette la predicibilità del sistema.

Inizialmente è stato detto che ogni DT è rappresentato da un Knowledge Graph (KG). All'interno di ogni DT c'è quindi una componente, detta Knowledge Graph Engine (KGE), che si occupa di gestire le relazioni con gli altri DT. Anche questa componente, così come l'SM comunica gli aggiornamenti al modulo di Storage.

Per concludere, l'Augmentation Engine (AE), si occupa di gestire tutto ciò che riguarda lo stato aumentato del DT. Similmente allo stato, lo stato aumentato è calcolato a partire dall'attuale stato del DT, dallo stato aumentato corrente e dall'evento ricevuto in ingresso.

3.2.2 Architettura della piattaforma

Nelle sezioni precedenti è descritta la possibile architettura interna di un Digital Twin; tuttavia, perchè il WoDT possa funzionare correttamente, sono necessari ulteriori moduli non riconducibili ai singoli DT, bensì alla piattaforma stessa. Nello specifico questi moduli sono: il Distributed Knowledge Graph Engine (DKGE), il Digital Twin Manager (DTM) e il Communication Layer (CL).

Il compito del DKGE è simile a quello del KGE. Il DKGE, a differenza di quest'ultimo, permette di visualizzare proprietà e relazioni di ogni DT facente parte del WoDT.

Il DTM si occupa invece di gestire l'intero ciclo di vita dei Digital Twin: dal momento della loro creazione, fino al momento della loro dismessa. Questo modulo comunica con i singoli DT attraverso il modulo della Management Interface, permettendo ad esempio di avviare l'esecuzione di nuovi Digital Twin, ovvero ordinando il binding e lo shadowing con il Physical Asset.

Per finire, il Communication Layer è il modulo che funge da collegamento tra il WoDT e le applicazioni esterne, come fa il Digital Adapter per i sin-

goli DT. Questo modulo consente alle applicazioni esterne di interagire con il WoDT attraverso i moduli precedenti, permettendo di eseguire operazioni che controllano il ciclo di vita del DT ed operazioni come interrogazioni sull'intero WoDT. Inoltre il CL permette di interagire con il Digital Adapter attreverso le operazioni definite dalla Management Interface.

3.3 WoDT e il livello applicazione

Nelle pagine precedenti è stato introdotto un possibile modello di architettura che rispetta i principi del Web of Digital Twin.

Vediamo ora come le applicazioni possono interagire con il WoDT ed i singoli Digital Twin che vi appartengono.

3.3.1 Interagire con il WoDT

Come già accennato in precedenza, il WoDT è ideato come una piattaforma che permette alle applicazioni di interagire con i Physical Asset attraverso i rispettivi Digital Twin, consentendo di eseguire operazioni come la creazione di nuovi DT, la lettura di proprietà e relazioni ed anche l'esecuzione di azioni sui Physical Asset, il tutto in un ambiente aperto e distribuito.

Le applicazioni interagiscono con il Digital Adapter mediante il Communication Layer, utilizzando le API fornite dalla Management Interface. Una delle necessità più comuni delle applicazioni è interrogare il sistema per conoscere lo stato dei Digital Twin, ovvero quali sono i valori delle proprietà, a quali altri DT puntano le relazioni e quali eventi sono avvenuti sul PA.

Il WoDT offre due diverse strategie per osservare i Digital Twin: la prima è la semplice query, mentre la seconda è il tracking.

Quando viene richiesta l'esecuzione di una query, il Digital Adapter inoltra la richiesta al Model Execution Engine, che a sua volta la comunica al Knowledge Graph Engine. Una volta eseguita la query, il risultato è trasmesso all'applicazione mediante il DA.

L'operazione di tracking funziona in maniera simile: quando si richiede il tracking, il DA comunica al Model Execution Engine la volontà dell'applicazione di osservare gli eventi registrati dal PA. Ogni qualvolta il PA rileverà il cambiamento, l'applicazione sarà informata attraverso il DA. Il tracking è molto utile in quanto permette di mantenere l'architettura event-driven anche a livello applicazione. Qualora l'applicazione non voglia più ricevere notifiche riguardo eventi accaduti sul PA, lo può comunicare al DA che procede annullando l'iscrizione attraverso il MEE.

Query e Tracking sono solo due delle possibili interazioni con i DT. Una richiesta comune è quella di eseguire operazioni sul PA, come ad esempio l'ac-

censione una lampadina attraverso il suo Digital Twin. Il DT della lampadina dovrà esporre mediante la Management Interface, una API che permetta di eseguire un comando sull'oggetto fisico, in questo caso l'accensione (o lo spegnimento). Va sottolineato che l'azione non cambia lo stato del DT, ciò che succede è invece questo: l'azione viene propagata al PA che la esegue, generando a sua volta un evento che è gestito dal sistema come un normale evento e_{PA}. Questa dinamica sarà meglio approfondita in seguito parlando del caso di studio pratico.

Il Digital Adapter consente di comunicare con i singoli DT, ma come visto poco fa, non è l'unico modo di interagire con il WoDT. Attraverso il Communication Layer è infatti possibile interagire anche con il DKGE per eseguire interrogazioni sull'intero grafo, e con il DT Manager per eseguire azioni sul ciclo di vita dei Digital Twin.

3.3.2 WoDT e sistemi multiagente

Per come è strutturato il Web of Digital Twin, non esiste un'unica soluzione all'architettura del livello applicazione. Una delle soluzioni comuni quando si parla in senso generico di sistemi di Digital Twin è l'uso di sistemi multiagente.

Nell'articolo riguardante il WoDT [6] sono presentate due diverse proposte. La prima è la più semplice e consiste nell'associare a ciascun Digital Twin un agente. L'agente comunica con il DT attraverso il Digital Adapter, ciò gli permette di conoscere le informazioni riguardanti lo stato del DT e al tempo stesso di eseguire azioni sul PA secondo le modalità viste in precedenza. La corrispondenza tra agenti e DT non è univoca, gli agenti direttamente associati al DT sono infatti solo una piccola parte del sistema. Essi fungono da interfaccia per gli altri agenti che vogliono comunicare con il DT, traducendo l'interfaccia esposta dal Digital Adapter ad un livello di astrazione più alto così da agevolare le operazioni precedentemente descritte. Gli agenti comunicano quindi tra loro utilizzando appositi protocolli di comunicazione, così facendo ogni agente può comunicare con ogni DT a seconda dell'occorrenza.

La seconda proposta consiste nell'associare ad ogni DT, non più direttamente un agente, bensì un artefatto. Gli agenti si interfacciano quindi all'artefatto corrispondente al DT con cui vogliono comunicare e, come nel caso precedente, possono anche comunicare tra loro attraverso opportuni protocolli di comunicazione.

Le architetture di WoDT e DT proposte limitano al minimo le dipedenze tra la piattaforma di Digital Twin ed il livello applicazione. Così facendo è possibile modificare l'architettura del DT a proprio piacere purché esso continui ad esporre la stessa interfaccia, tutto questo senza che il livello applicazione noti alcun cambiamento.

Capitolo 4

Prototipazione di un sistema WoDT attraverso Azure DT

Nei capitoli precedenti sono state descritte le caratteristiche dei Digital Twin e le idee chiave del Web of Digital Twin. Uno dei punti principali di questa tesi è la sperimentazione di ciò che è stato presentato in precedenza, questo include la progettazione e lo sviluppo di un sistema di Digital Twin applicato ad un caso di studio pratico, con tutte le caratteristiche che il WoDT comporta: dalla dinamicità del sistema alle relazioni tra le entità che vi appartengono.

Nelle pagine successive verrà esposto il caso studiato e saranno analizzate le soluzioni proposte per la sua risoluzione ed alcuni esempi di funzionamento del sistema. Saranno inoltre presentate alcune architetture alternative ed i motivi per cui queste sono state scartate.

4.1 Azure DT applicato al WoDT: limiti e similitudini

Per la realizzazione dell'ecosistema di Digital Twin che sarà descritto nelle pagine successive, si è scelto di utilizzare la piattaforma Azure Digital Twin in quanto essa è in grado di soddisfare tutti o quasi i principi indicati dal Web of Digital Twin. Di seguito sono mostrati, oltre alle feature supportate, anche i limiti che Azure Digital Twin impone allo sviluppo di un sistema WoDT.

Per quanto riguarda la sincronizzazione tra Physical Asset e Digital Twin, entrambe le proposte adottano strategie simili ma non uguali. In entrambi i casi è il Physical Asset che a fronte di un cambiamento o di un evento, comunica al Digital Twin il proprio stato aggiornato. Nel WoDT questa comunicazione avviene in maniera diretta da Physical Asset a Digital Twin attraverso il Physical Asset Adapter (PAA). Azure Digital Twin supporta questa procedura di

sincronizzazione, che può essere effettuata attraverso l'uso delle REST API precedentemente descritte, tuttavia questa non rappresenta l'unica modalità tramite cui può essere effettuato lo shadowing. Ogni applicazione autorizzata a comunicare con l'istanza di Azure può infatti aggiornare il valore delle proprietà di qualsiasi Digital Twin. La modalità di sincronizzazione attraverso l'IoT Hub precedentemente descritta adotta proprio questo principio.

Come detto in precedenza, due degli aspetti chiave del WoDT sono le relazioni e la dinamicità del sistema. Anche in questo caso la soluzione proposta da Azure è compatibile con i requisiti del WoDT: Azure Digital Twin permette di definire relazioni tra i DT, inoltre consente di creare nuove relazioni ed eliminarne di esistenti dinamicamente, consentendo anche di cambiare il target della relazione a run-time. Tuttavia, mentre le relazioni nel WoDT riflettono sempre i legami che i Physical Asset possiedono nel mondo fisico, all'interno di Azure Digital Twin non sono posti vincoli sul quando e sul come usare le relazioni. Volendo possono quindi essere create relazioni che non rispecchiano il mondo fisico, è quindi compito di chi progetta il sistema evitare che ciò non accada.

Entrambe le proposte offrono un insieme di funzionalità per interagire con la piattaforma, permettendo di effettuare sia operazioni sul sistema, come la creazione o l'eliminazione di Digital Twin, sia operazioni sui singoli Digital Twin, come la modifica delle relazioni. La creazione e l'eliminazione di DT a run-time è quindi un altro dei requisiti di WoDT che Azure Digital Twin rispetta, ma il concetto di dinamicità offerto da questa piattaforma non si limita a questo. Sempre attraverso l'uso delle API è infatti possibile definire a run-time nuovi modelli o modificare quelli già esistenti. Questa funzionalità permette una grande flessibilità del sistema, consentendo di aggiungere nuove feature anche in fasi successive al rilascio.

Un altra delle caratteristiche di un sistema di Digital Twin è la gestione dello storico. Azure Digital Twin si avvale a questo scopo del servizio Azure Data Explorer, mentre il WoDT non pone vincoli tecnologici alla realizzazione di questa sottoparte del sistema, limitandosi a definire l'esistenza di una componente che si occupa di interagire con lo storico. In entrambi i casi, lo storico viene aggiornato ogni qualvolta lo stato del DT cambia e può essere consultato attraverso l'uso di query. Il WoDT offre inoltre una funzionalità aggiuntiva, ovvero il tracking, che può essere emulato anche all'interno della piattaforma Azure Digital Twin utilizzando le Azure Function e le telemetrie.

Azure Digital Twin sembra essere quindi una buona soluzione allo sviluppo di complessi sistemi di Digital Twin. Utilizzando questa piattaforma non ci dovremmo preoccupare personalmente di problemi complessi come l'autenticazione o la scalabilità, tuttavia al tempo stesso dovremo pagare il servizio offerto dalla piattaforma, che a seconda della complessità del nostro sistema

può raggiungere cifre più o meno elevate.

4.2 Introduzione al caso di studio

Il caso di studio scelto per la prototipazione di un sistema WoDT è quello del Button-Led. Questo caso di studio, pur non essendo troppo complesso, richiede che siano modellati tutti gli aspetti principali del Web of Digital Twin.

L'idea è che alla pressione del pulsante, la luce ad esso collegato si accenda, mentre al rilascio torni ad essere nuovamente spenta. Nel sistema sono quindi presenti sia dei sensori, ovvero dei dispositivi di input, che degli attuatori sui quali eseguire azioni.

Il sistema, oltre a gestire le operazioni di shadowing, deve reagire agli eventi di pressione del pulsante e inviare azioni alle luci. Inoltre il sistema deve permettere alle applicazioni esterne di interagire con i Digital Twin ed i relativi Physical Asset, consentendo di effettuare operazioni come il tracking dei DT o l'esecuzione di azioni sui PA.

La soluzione inoltre non deve essere specifica per la risoluzione di questo determinato problema ma deve essere possibile adattarla a qualsiasi altro contesto applicativo.

Al problema del Button-Led appena descritto si è scelto di aggiungere alcune feature per rendere il problema un po' più complesso ed evidenziare alcuni aspetti importanti del Web of Digital Twin, rendendo inoltre il suo studio più interessante e realistico:

- Il problema preso in esame considera il caso in cui più pulsanti e più luci sono presenti contemporaneamente all'interno del sistema.

- I pulsanti possono anche non essere associati ad alcuna luce e le luci possono non essere comandate ad alcun pulsante. Per rendere il sistema più fedele e applicabile alla realtà, una luce deve poter essere comandata anche da più di un pulsante.

- Viene introdotto il concetto di *stanza*: pulsanti e luci sono collegati tramite una relazione ad al più una stanza. Aggiungere l'entità della stanza permette di eseguire interrogazioni del tipo "seleziona tutte le luci all'interno di una determinata stanza". A questa query può seguire l'azione di accendere tutte le luci restituite.

- La relazione tra pulsante e luce non deve essere statica. Uno dei principi chiave del Web of Digital Twin è infatti la dinamicità: si vuole fare in modo che le relazioni possano essere modificate anche a run-time. Allo

stesso modo deve essere possibile creare dinamicamente e in maniera automatica i Digital Twin corrispondenti ai nuovi dispositivi collegati.

- La soluzione deve includere un'applicazione che attraverso un'interfaccia permetta di gestire manualmente il sistema. L'applicazione deve mostrare i dispositivi presenti all'interno del sistema, quindi pulsanti, luci e stanze, e deve essere possibile creare ed eliminare in fase di esecuzione le relazioni tra essi, come detto nel punto precedente.

La soluzione deve includere quindi tutti gli strati del sistema (fisico e digitale) e le comunicazioni tra essi.

- A livello fisico deve gestire la comunicazione tra i Physical Asset ed i relativi Digital Twin. La comunicazione può avvenire in entrambe le direzioni, ad esempio da PA a DT per gestire il binding e lo shadowing e da DT a PA per gestire l'esecuzione di azioni sul PA.
- A livello della piattaforma WoDT vera e propria, deve gestire il binding e lo shadowing tra PA e DT, le relazioni tra i vari DT e deve fornire delle interfacce che permettano alle applicazioni esterne di interagire con i DT.

4.3 L'architettura proposta

Prima di analizzare nel dettaglio le varie componenti del sistema, è utile fare un'introduzione della sua architettura (figura 4.1), in modo da fornire un quadro generale.

Per semplificare la visione del sistema, possiamo pensarlo come costituito da tre strati principali:

- il PA Layer (strato fisico)
- il Device Manager
- il DT Layer

Lo strato fisico è costituito dai Physical Asset, nel caso di studio quindi i pulsanti e le luci. I Physical Asset comunicano con il Device Manager inviando e ricevendo informazioni. Approfondiremo più avanti le modalità di comunicazione ed il formato dei messaggi, per ora è sufficiente sapere che la comunicazione avviene utilizzando il protocollo MQTT.

Il Device Manager fa da intermediario tra il PA Layer e il DT Layer, ed è costituito da due componenti principali: l'MQTT Broker e il DT Manager.

Come già accennato, il Broker MQTT si occupa della comunicazione con i Physical Asset, ma al tempo stesso permette alle applicazioni esterne di interagire con i Digital Twin ed i relativi PA.

La gestione vera e propria dei Digital Twin è invece affidata al DT Manager. Sulla base delle richieste ricevute dal Broker MQTT, il DT Manager procede eseguendo sia operazioni di scrittura come lo shadowing, sia operazioni di lettura come ad esempio le query.

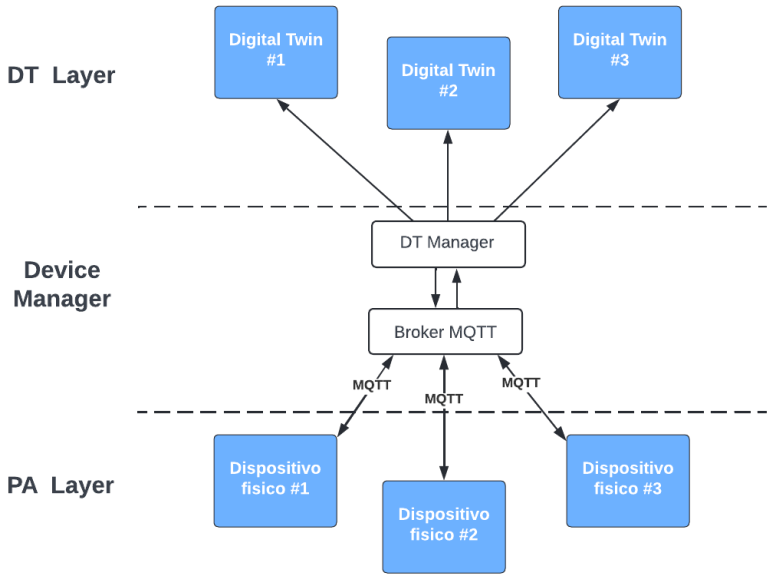


Figura 4.1: schema generale dell'architettura proposta

Mentre nell'immagine 4.1 è mostrata l'architettura generica senza un particolare focus sulle tecnologie utilizzate, nell'immagine in figura 4.2 è mostrata l'architettura specifica del sistema progettato, con l'indicazione delle tecnologie effettivamente utilizzate.

Per quanto riguarda il Broker MQTT, si è scelto di utilizzare *Vertx*, mentre per gestire il DT Layer si è fatto affidamento sulla piattaforma Azure Digital Twin, ampiamente discussa nei capitoli precedenti.

Il DT Manager comunica con l'istanza di Azure Digital Twin attraverso l'uso delle API fornite sotto forma di SDK per il linguaggio di programmazione *Java*.

Nelle pagine che seguono sono descritte nel dettaglio le singole componenti, come sono state progettate e come esse comunicano tra loro per consentire il corretto funzionamento del sistema.

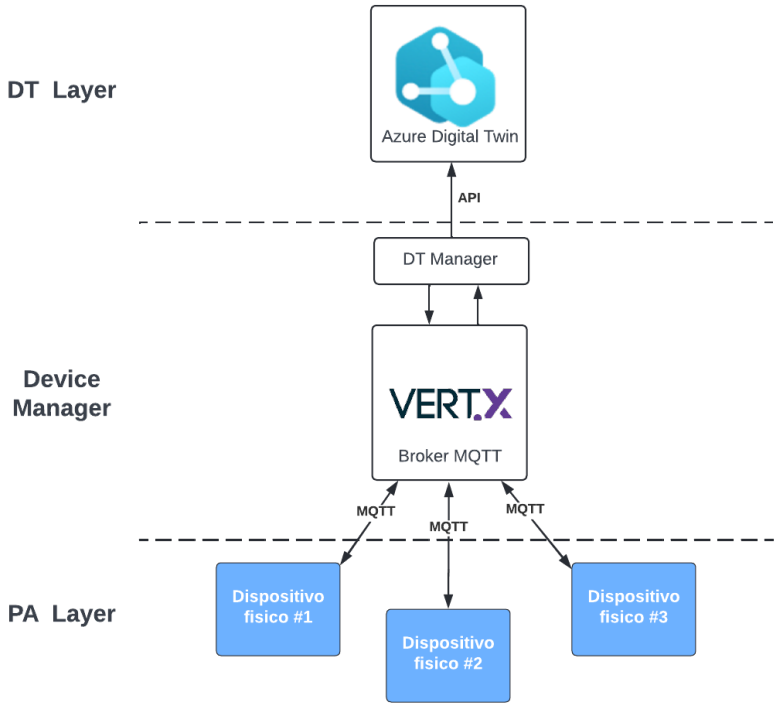


Figura 4.2: Schema dettagliato dell'architettura progettata

4.3.1 Logica dei Physical Asset

Generalmente quando si parla di Physical Asset, si sta considerando una categoria molto vasta di risorse che tra loro possono avere poco o nulla in comune. Nonostante il caso di studio comprendesse un numero molto limitato di classi di Physical Asset, si è cercato di realizzare un'architettura che fosse il più possibile flessibile, in modo da renderla espandibile anche ad altri domini applicativi.

Quando un dispositivo si accende, la prima operazione che svolge è collegarsi alla rete WiFi. Una volta che la connessione è avvenuta con successo, procede quindi a creare una connessione con il Broker MQTT.

Non appena è stabilita la connessione con il Broker, il Physical Asset pubblica un messaggio sul canale *createAndBind*, comunicando l'id del Digital Twin associato e l'id del modello di cui il DT è istanza. Per tutte le fasi della comunicazione si è scelto di mantenere JSON come formato per lo scambio dei messaggi, in modo da rendere la comunicazione comprensibile anche ad un operatore umano.

Per completare la fase di inizializzazione del sistema, il dispositivo effettua una prima operazione di shadowing, pubblicando un messaggio sul canale *sha-*

dowing. Il messaggio contiene le informazioni riguardanti lo stato dell'intero sistema, in modo da fornire alla controparte virtuale uno stato di partenza consistente con quello della componente fisica. Pulsanti e luci comunicheranno rispettivamente i valori delle proprietà *isPressed* e *isOn*.

Con questa operazione di shadowing ha fine la fase di inizializzazione: da ora in poi ogni dispositivo adotta il proprio comportamento, diverso da quello degli altri PA, fatta eccezione per l'operazione di keepAlive.

Può capitare per diversi motivi che la connessione tra il Broker ed i dispositivi si interrompa e mentre la vita dell'oggetto fisico prosegue, la sua controparte virtuale rimane nell'ultimo stato conosciuto prima della disconnessione. È quindi utile verificare periodicamente la validità della connessione. Questa operazione può essere svolta in diversi modi, quello adottato da questa architettura è il seguente: i dispositivi fisici periodicamente pubblicano un messaggio sul canale *keepAlive*, comunicando al Broker che sono ancora attivi e funzionanti, nonostante magari non stiano mandando alcun messaggio. Quello che accade all'arrivo del messaggio sarà approfondito nella sezione relativa al Broker, per ora è sufficiente essere a conoscenza dell'esistenza di questa feature.

Logica dei pulsanti

Il pulsante adotta un modello interrupt-driven: quando viene rilevato un cambiamento sul pin di input, il dispositivo comunica sul canale *shadowing* il proprio nuovo stato. Se il pulsante è stato premuto, il valore della proprietà *isPressed* sarà *True*, al contrario, se è stato rilasciato sarà *False*. Come vedremo più nel dettaglio in seguito, il pulsante fisico non è a conoscenza dell'esistenza di altri dispositivi, ma si limita a comunicare il proprio stato al Broker. Il pulsante quindi non sa chi sta comandando e potrebbe anche non essere collegato ad alcuna luce e lo stesso è valido anche a parti inverse. Nella logica adottata, le relazioni non sono memorizzate a livello di dispositivi fisici, bensì a livello di DT.

Logica delle luci

Anche la luce, come detto poco fa, svolge le operazioni iniziali comuni a tutti i PA, ovvero: connessione alla rete WiFi, connessione al Broker MQTT, binding e shadowing. Essendo tuttavia la luce un attuatore, la sua logica da questo punto in avanti è opposta a quella del pulsante. Dopo avere eseguito l'operazione di Binding, il dispositivo si iscrive al canale *action/dt-id*, mettendosi così in ascolto di eventuali comandi provenienti dal Broker.

Quando la luce riceve un messaggio in questo canale, procede deserializzando il corpo del messaggio e legge l'operazione corrispondente alla proprietà *op*: se il valore è uguale a *on*, la luce si accende, in alternativa se è uguale a

off, si spegne. In entrambi i casi la luce effettua successivamente l'operazione di shadowing, indicando il valore aggiornato della proprietà *isOn*.

Questo è un passaggio fondamentale nella logica dei Digital Twin: le applicazioni esterne non possono modificare direttamente i valori delle proprietà dei Digital Twin. Solo il Device Manager può eseguire questa operazione e solo a seguito dell'operazione di shadowing da parte del Physical Asset. Questo passaggio potrebbe sembrare superfluo, tuttavia non lo è ed impedisce al sistema di entrare in uno stato di inconsistenza. Si prenda per esempio il caso in cui il PA della luce si disconnette per qualche motivo dal Broker MQTT. Se il messaggio non arriva al PA, esso di conseguenza non può reagire, e la luce non può accendersi (o spegnersi). Se l'applicazione che ha comandato l'operazione modifica l'istanza del DT, pensando che l'operazione sia andata a buon fine, commette un errore, in quanto la componente virtuale non rappresenterà più una copia valida di quella fisica.

In figura 4.3 è mostrato come, alla pressione del pulsante, la luce si accende. Come appena detto, la comunicazione non è diretta pulsante-luce ma passa attraverso il broker.

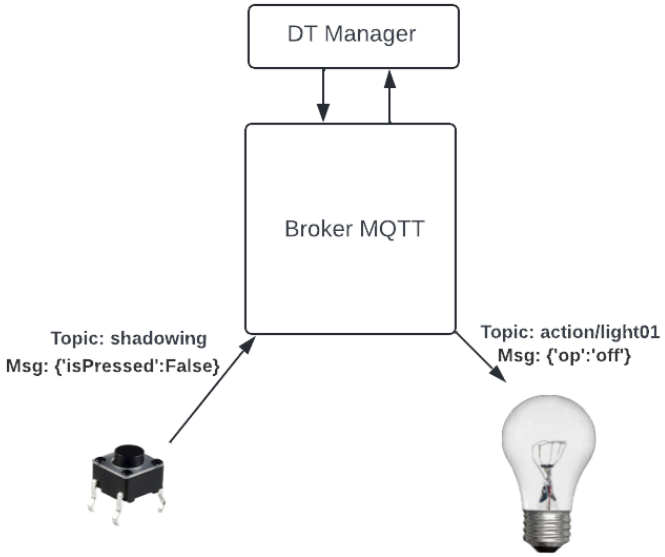


Figura 4.3: Comunicazione indiretta tra un pulsante e una luce

Emulazione dei dispositivi

La logica descritta in precedenza permette di creare applicazioni che simulino il funzionamento dei dispositivi fisici senza la necessità di ulteriori compo-

nenti elettroniche come ESP, luci o pulsanti. Un singolo dispositivo è infatti in grado di simulare il funzionamento di decine di Physical Asset.

Il codice che segue (listato 4.1) mostra una parte del codice che permette di emulare il funzionamento di un Physical Asset attraverso uno script Python. Nello specifico, il codice è relativo ad un pulsante ma le operazioni iniziali sono valide anche per gli altri Physical Asset.

Come si può notare sono presenti tutte le operazioni descritte in precedenza: dopo che la connessione è stata istanziata, viene pubblicato un messaggio sul canale *createAndBind* contenente l'id del Digital Twin ed il modello a cui appartiene; successivamente è effettuato lo shadowing indicando il valore delle proprietà da sincronizzare. In questo caso, trattandosi dell'esempio specifico di un pulsante, nel payload del primo messaggio è indicato il modello *dtmi:progettotesi:button;1*, mentre nello shadowing è specificata la proprietà *isPressed*.

Il codice include anche l'operazione di keep-alive: ogni 50 secondi il dispositivo emulato pubblica un messaggio vuoto sul canale *keepAlive* in modo da mantenere la connessione attiva ed allo stesso tempo comunicare la propria presenza al Broker.

```
def keep_alive():
    client.publish("keepAlive", "", 0);
    threading.Timer(50.0, keep_alive).start()

client.connect("192.168.178.104", 1883, 100)
client.loop_start()

threading.Timer(50.0, keep_alive).start()

payload = {"device-id":"button-1",
           "model-id":"dtmi:progettotesi:button;1"}
client.publish("createAndBind", json.dumps(payload), 0)

payload = {'isPressed':False}
client.publish("shadowing", json.dumps(payload), 0)
```

Listato 4.1: Emulazione di un Physical Asset attraverso Python

Mentre il comportamente del pulsante è molto semplice: lo shadowing è effettuato direttamente a seguito di un input da parte dell'utente, può essere interessante analizzare come si comporta la luce a seguito della ricezione di un messaggio (listato 4.2).

All'arrivo di un messaggio è verificato il topic: se il topic corrisponde ad *"action/light-01"*, viene letto il valore della proprietà *op* e la luce viene accesa

o spenta di conseguenza, simulandolo attraverso un messaggio nell'interfaccia utente. In seguito è effettuata l'operazione di shadowing indicando il nuovo valore della proprietà.

```
client.subscribe("action/light-01")

def on_message(client, userdata, msg):
    if msg.topic == "action/light-01":
        payload = json.loads(msg.payload.decode())
        if('op' in payload):
            global lightOn
            if payload['op'] == 'on':
                lightOn = True
            if payload['op'] == 'off':
                lightOn = False

        print("lightOn: " + 'on' if lightOn else 'off')
        shadowingInfo = {'isOn': lightOn}
        client.publish("shadowing", json.dumps(shadowingInfo), 0);
```

Listato 4.2: Comportamento emulato di una luce

4.3.2 Il Broker MQTT

Insieme al DT Manager, il Broker MQTT è una delle due componenti principali che costituiscono il Device Manager. Il Broker gestisce sia le comunicazioni con i dispositivi, che siano essi reali o simulati, sia con le applicazioni esterne che vogliono interagire con la piattaforma.

Per la comunicazione si è scelto di utilizzare il procollo MQTT in quanto risulta essere uno dei migliori procolli per la comunicazione tra dispositivi embedded. MQTT rispetto ad altri protocolli simili, come ad esempio HTTP, richiede un minore consumo in termini di energia, il che si traduce in una vita più lunga per i dispositivi embedded che lo adottano.

Il Broker MQTT è stato sviluppato in Java utilizzando *Vert.x*, partendo da un Server MQTT e adattandolo alle specifiche richieste. Il Broker lavora a stretto contatto con il DT Manager per tradurre i messaggi ricevuti in operazioni da effettuare sul sistema: dalla creazione di un nuovo Digital Twin allo shadowing del Physical Asset. Come descritto nelle pagine successive, il Broker gestisce inoltre le iscrizioni ai canali da parte dei client ed offre un meccanismo di autenticazione e validazione delle connessioni.

Gestione delle iscrizioni

Quando il Broker riceve un messaggio di tipo *subscribe*, aggiunge l'utente che ha inviato il messaggio alla lista degli iscritti al topic indicato.

Ogni qualvolta il Broker riceve un messaggio di tipo *publish*, scorre la lista delle iscrizioni alla ricerca di un topic che inizi con la stringa indicata o che sia uguale ad essa ed inoltra il messaggio appena ricevuto ai dispositivi iscritti al canale trovato (listato 4.3).

```
public void publish(String topic, JsonObject payload) {
   for (String s : subscriptions.keySet()) {
      if(topic.startsWith(s + "/") || topic.equals(s)) {
         for (MqttEndpoint e : subscriptions.get(s)) {
            e.publish(topic, payload.toBuffer(),
                MqttQoS.AT_LEAST_ONCE, false, false);
         }
      }
    }
}
```

Listato 4.3: Codice che mostra come i messaggi sono inoltrati agli iscritti di un topic

Il broker non pone limiti alle iscrizioni: qualsiasi applicazione o dispositivo può iscriversi a qualsiasi topic senza restrizioni. In generale possiamo classificare le iscrizioni in due categorie: le applicazioni esterne che si iscrivono al canale di shadowing in modo da ricevere i cambiamenti di stato dei dispositivi, e i Physical Asset che si iscrivono al canale *action/dt-id* in modo da ricevere le azioni richieste dal livello applicazione. Nel listato 4.2 è mostrato un esempio di iscrizione del secondo tipo e la relativa gestione lato Physical Asset.

Se un dispositivo vuole dare alle applicazioni esterne la possibilità di iscriversi al canale di shadowing specifico a se stesso, può pubblicare i messaggi sul topic *shadowing/device-id* anziché sul generico canale di *shadowing*. Questa seconda operazione è utile qualora si volesse fornire la feature del tracking: un'applicazione si iscrive al canale di shadowing di un determinato PA in modo da essere a conoscenza di ogni suo cambiamento, permettendole così di agire di conseguenza.

Gestione delle autenticazioni e sicurezza

Quando un dispositivo richiede di connettersi al Broker MQTT, deve provare la propria identità fornendo uno username ed una password.

Lo username è univoco ed in fase di binding è associato all'id del Digital Twin. Quando il Broker riceve una richiesta di shadowing, legge dalla mappa

dei dispositivi l'id del DT corrispondente allo username. Così facendo si scongiura la possibilità che una connessione malevola possa eseguire lo shadowing di un qualsiasi Digital Twin, in quanto soltanto lo username collegato a quel DT può eseguire lo shadowing, e quest'ultimo è protetto da password.

All'interno del sistema le password non sono memorizzate in chiaro ma codificate attraverso una codifica one-way: SHA3-256.

Attualmente il sistema non contempla la gestione di dispositivi malevoli che si fingono pulsanti o luci pur non essendo tali: è infatti possibile, come mostrato poco fa, creare degli emulatori che si fingano Physical Asset ed eseguano operazioni di binding o shadowing.

Gestione dei canali e formato dei messaggi

Come detto in precedenza, i messaggi seguono il formato JSON, in modo da rendere la comunicazione comprensibile anche da un operatore umano. Gli unici messaggi che fanno eccezione sono quelli inviati sul canale di *binding*. Questi messaggi, avendo il solo scopo di segnalare la propria esistenza al Broker, non hanno infatti nessun payload.

Gli altri due canali comuni su cui comunicano i dipositivi sono: *createAndBinding* e *shadowing* (o *shadowing/device-id*). Il formato dei messaggi pubblicati sul primo canale è mostrato di seguito (listato 4.4).

```
{
    "device-id":<<device-id>>,
    "model-id":<<model-id>>
}
```

Listato 4.4: formato del messaggio di binding

Mentre i messaggi pubblicati sul canale *createAndBinding* hanno tutti lo stesso insieme di proprietà (*device-id* e *model-id*), quelli pubblicati sul canale di *shadowing* variano in base al Physical Asset e alle proprietà che il PA richiede di aggiornare (listato 4.5). Allo stesso modo, anche i messaggi pubblicati sul canale *action/dt-id* dipendono dal tipo di Physical Asset a cui sono destinati.

```
{
    <<property-1>>:<<value>>,
    <<property-2>>:<<value>>,
    ...
}
```

Listato 4.5: formato del messaggio di shadowing

4.3.3 DT Manager

Il DT Manager è la componente che si occupa di aggiornare lo stato dei Digital Twin presenti sulla piattaforma sulla base delle informazioni ricevute dal Broker MQTT. Molte delle operazioni svolte dal DT Manager sono effettuate su un thread diverso da quello che gestisce il Broker MQTT, questo perchè alcune operazioni, come ad esempio query complesse, potrebbero richiedere tempo per essere eseguite e rallenterebbero pertanto il lavoro del Broker se effettuate sullo stesso thread.

I Digital Twin veri e propri risiedono in un'istanza di Azure Digital Twin. Si è scelto di utilizzare questa piattaforma per via dei numerosi vantaggi precedentemente elencati, tra cui ed esempio la possibilità di creare relazioni tra Digital Twin.

Il DT Manager è quindi connesso a una specifica istanza di Azure Digital Twin e quando l'applicazione viene avviata, la prima operazione che effettua in seguito alla connessione è sincronizzarsi con l'istanza indicata, richiedendo la lista di tutti i DT presenti e delle relazioni che intercorrono tra essi. In questo caso Azure Digital Twin agisce quindi da layer di persistenza consentendo, nel caso ci siano malfunzionamenti con l'applicazione, di terminare e riavviare l'esecuzione senza perdere la sincronizzazione con l'istanza di Azure associata. Si è preferito utilizzare Azure Digital Twin come layer di persistenza anziché un semplice database locale, in modo da ridurre la ridondanza dei dati.

Per interfacciarsi con l'istanza di Azure Digital Twin, il DT Manager utilizza l'SDK di Azure disponibile per il linguaggio Java. Tra le operazioni che il DT Manager permette di effettuare ci sono: la creazione di nuovi Digital Twin, la creazione e l'eliminazione di relazioni tra DT, lo shadowing, la reazione ai cambiamenti di stato e la gestione del keep-alive. Vediamo ora alcune di queste operazioni più nel dettaglio.

Creazione e Shadowing

Il DT Manager fornisce la funzione `createDT(String dtId, String modelId)` che permette la creazione di nuovi Digital Twin. La funzione prende in input l'id del DT da creare ed il modello di cui esso deve essere istanza.

La prima operazione che il DT Manager effettua è verificare che non esista già un DT con il dato id. Se così fosse, annulla l'operazione di creazione. Nel caso invece l'id non sia in uso, procede creando l'istanza del nuovo Digital Twin. In questa fase la proprietà *alive* del DT viene impostata a *False* in quanto l'operazione di shadowing non è ancora stata effettuata.

La funzione `shadowDT(String dtId, JsonObject payload)` richiede di indicare l'id del Digital Twin di cui si vuole effettuare lo shadowing e l'insieme di proprietà da aggiornare con i rispettivi valori. Nello specifico, questo secondo

parametro, corrisponde al payload del messaggio ricevuto dal Broker MQTT ed adotta pertanto il formato JSON. Nel listato che segue (4.6) è mostrata parte del corpo della funzione che permette di aggiornare i valori delle proprietà richiesti.

```
final JsonPatchDocument jsonPatchDocument = new JsonPatchDocument();

payload.forEach(e -> {
   jsonPatchDocument.appendReplace("/" + e.getKey(), e.getValue());
});

dtClient.updateDigitalTwin(dtId, jsonPatchDocument);
```

Listato 4.6: Operazione di aggiornamento del Digital Twin

A seguito dell'operazione di aggiornamento è eseguita la reazione al cambiamento di stato. In questo specifico caso di studio, l'unica classe di Digital Twin che reagisce ai cambiamenti è quella del pulsante. Le luci infatti sono dispositivi puramente passivi, ma non è da escludere che in una visione più ampia possano adottare anche loro una propria reazione. Ad ogni modo, per come è strutturata l'applicazione, questa funzione è facilmente implementabile.

Inoltre in questo caso, tutti i Digital Twin appartententi alla stessa classe adottano la stessa reazione, ma nulla vieta che alcuni DT possano avere alcune reazioni mentre altri reazioni diverse.

Ciò che succede quando viene invocato il metodo `react(String dtId, JsonObject payload)` passando come *dtId*, l'id di un pulsante è questo:

- attraverso la lista delle relazioni è identificata la luce connessa al pulsante;
- si controlla se il valore di *isPressed* è True o False, ovvero se l'operazione da eseguire è l'accensione o lo spegnimento;
- si richiede al broker di pubblicare un messaggio sul canale *action/light-id* contentente l'azione da eseguire (on/off) sulla luce trovata.

È anche possibile che il pulsante non sia connesso ad alcuna luce. In tal caso la ricerca della relazione non restituirà nulla e al broker non sarà richiesto di eseguire alcuna pubblicazione.

Keep-alive

Come detto poco fa, il broker gestisce un topic specifico dedicato alla funzionalità del keep-alive. Quando un messaggio è ricevuto su questo canale, il

broker invoca la funzione `keepAlive` del DT Manager, indicando come parametro l'id del dispositivo che ha inviato il messaggio. Il DT Manager procede rimuovendo il DT indicato dalla lista contenente i DT inattivi e aggiorna il valore della proprietà *alive* (figura 4.7) del DT al valore *True*. Infine, il DT Manager aggiorna l'orario dell'ultima comunicazione ricevuta dal PA.

Questa procedura è effettuata non solo quando il broker riceve un messaggio sul topic *keepAlive*, ma anche quando viene ricevuto un messaggio sul canale *shadowing*.

Il DT Manager al suo interno esegue una routine di controllo delle connessioni. Periodicamente il DT Manager controlla quanto tempo è passato dall'ultima comunicazione di ogni dispositivo e se questa differenza è superiore al *keep-alive time*, aggiunge il dispositivo alla lista dei dispositivi disconnessi e imposta la proprietà *alive* del Digital Twin a False.

```
{
    "@type": "Property",
    "name": "alive",
    "schema": "boolean"
}
```

Listato 4.7: Proprietà alive dei Digital Twin

Gestione delle relazioni

Una delle prime operazioni che il DT Manager effettua all'avvio è la sincronizzazione delle relazioni. Le relazioni sono poi conservate dal DT Manager attraverso una lista. Adottando questa strategia, le operazioni sui Digital Twin risultano essere più veloci in quanto non è necessario, interrogare ripetutamente l'istanza di Azure.

Attraverso il DT Manager, è possibile creare ed eliminare dinamicamente le relazioni tra i Digital Twin. Il DT Manager fornisce inoltre delle funzioni *get* che permettono di ottenere le relazioni che soddisfano determinate condizioni, per esempio tutte le relazioni che hanno una determinata sorgente o un determinato destinatario, con la possibilità di specificare anche il nome della relazione che si sta cercando.

Attualmente la creazione e l'eliminazione delle relazioni è possibile solamente attraverso l'interfaccia fornita dall'applicazione, tuttavia rimane aperta la possibilità di gestire anche le relazioni attraverso il Broker MQTT, e quindi tramite le applicazioni esterne. Ad esempio è possibile creare il canale *createRelationship* che prende in input gli id di sorgente e destinatario e il nome della relazione e gestirlo in maniera simile al canale *createAndBinding*.

4.4 Esempio di funzionamento del sistema

Dopo aver descritto l'architettura del sistema progettato, si procede analizzando alcuni casi d'uso per capire come le componenti si relazionano tra loro per permettere di interagire con i Digital Twin del sistema.

L'amministratore del sistema interagisce con il sistema attraverso l'applicazione mostrata di seguito (figura 4.4). Questa applicazione interagisce direttamente con il DT Manager per eseguire operazioni sulle istanze dei Digital Twin, come ad esempio la gestione delle relazioni tra pulsanti e luci e l'appartenenza dei DT alle stanze.

Come detto in precedenza, all'avvio dell'applicazione, il DT Manager scarica dall'istanza di Azure Digital Twin la lista delle istanze e le relazioni che intercorrono tra esse. I Digital Twin trovati sono quindi mostrati in tre liste, ognuna rappresentante una diversa classe. Da sinistra verso destra troviamo: i pulsanti, le luci e le stanze.

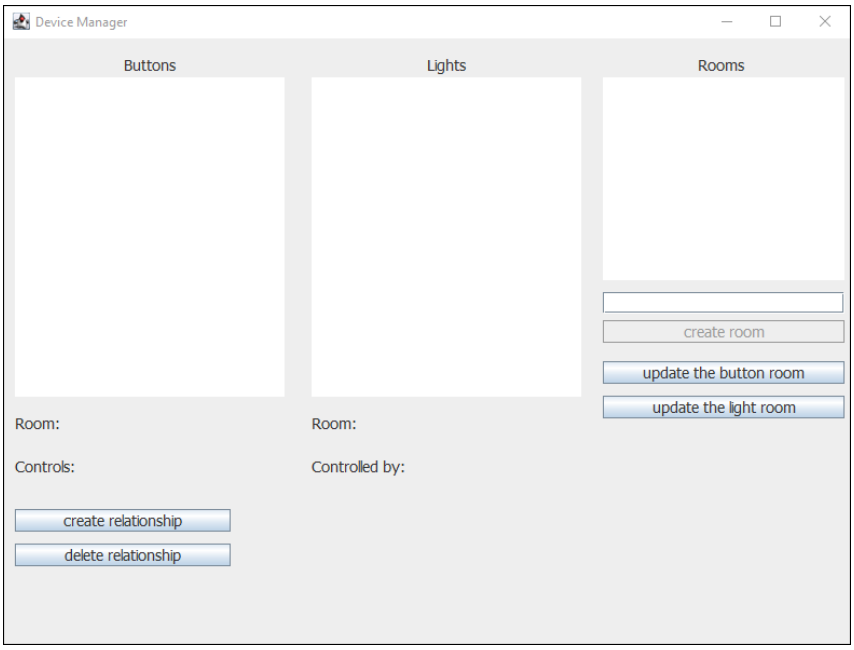


Figura 4.4: Interfaccia dell'applicazione che permette di gestire il sistema

Durante la spiegazione del funzionamento sarà utilizzata anche della vista offerta dalla piattaforma Azure Digital Twin Explorer per mostrare graficamente come le relazioni create attraverso l'applicazione si riflettono realmente all'interno del sistema.

4.4.1 Creazione e Binding di nuovi Digital Twin

Nel caso di partenza il sistema è vuoto, ovvero non contiene alcun Digital Twin. Avviando l'applicazione, essa procederà a creare un Broker MQTT e si connetterà all'istanza di Azure DT indicata attraverso il DT Manager.

Non essendoci alcun Digital Twin, le tre liste risulteranno vuote ma non appena il DT Manager si connetterà all'istanza di Azure sarà possibile creare manualmente nuovi Digital Twin della classe *room*. A differenza dei pulsanti e delle luci, le stanze richiedono di essere create manualmente da un amministratore del sistema in quanto non sono dotate di un proprio sistema embedded che si connette fisicamente all'applicazione, ma sono a tutti gli effetti dei contenitori per gli altri Digital Twin.

È possibile creare una nuova stanza attraverso il form dedicato: nell'esempio che segue è mostrata la creazione della stanza "cucina". Attraverso le API, il DT Manager ordinerà all'istanza di Azure di creare un nuovo Digital Twin appartenente alla classe *dtmi:progettotesi:room;1* con il nome indicato dal form. Si può verificare l'avvenuta creazione sia attraverso l'applicazione (la lista delle stanze si popolerà con l'elemento creato) (figura 4.5), sia attraverso l'Explorer di Azure.

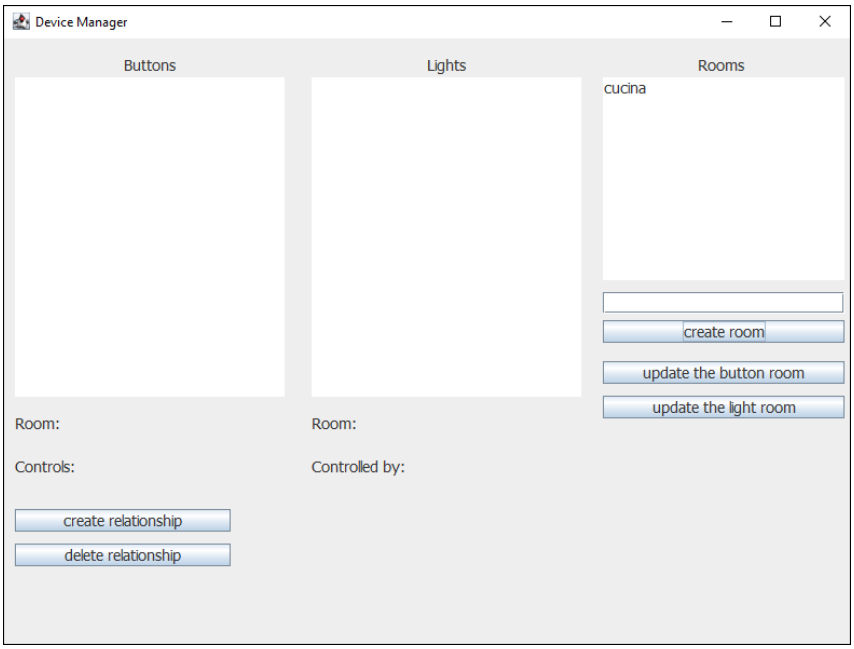


Figura 4.5: visualizzazione della stanza appena creata.

La creazione delle stanze può avvenire sia prima che dopo il binding degli altri Physical Asset, in quanto la creazione non comporta l'obbligo di creare

una relazione. In una qualsiasi fase dell'esecuzione del sistema potrebbe esserci una stanza che non contiene nessun Digital Twin.

Si procede ora con la creazione dei DT corrispondenti ai pulsanti e alle luci. A differenza di ciò che accadeva con le stanze, che devono essere create manualmente dall'amministratore, i pulsanti e le luci creano autonomamente i propri Digital Twin una volta accesi e connessi al broker MQTT. In questo caso sono state create tre istanze di ciascuna delle due classi in modo da poter osservare un caso più completo e realistico rispetto al semplice esempio con un pulsante ed una luce. Allo stesso tempo è stata creata una seconda stanza ("salotto") per contenere alcuni di questi Digital Twin. Nelle figura in seguito si può osservare l'evoluzione del sistema in seguito alla creazione dei nuovi dispositivi, come prima sia attraverso l'applicazione dell'amministratore, sia attraverso l'Azure DT Explorer (figura 4.6).

Figura 4.6: visualizzazione di tutti i DT appartenenti al sistema attraverso l'Azure Digital Twin Explorer

Il sistema contiene ora 8 Digital Twin:

- 3 pulsanti: button-01, button-02 e button-03;
- 3 luci: light-01, light-02 e light-03;
- 2 stanze: cucina e salotto.

Tutti i Digital Twin sono attualmente slegati tra loro, ma ciò non significa che essi non rispecchiano lo stato dell'oggetto reale. I Digital Twin effettuano infatti lo shadowing, ad esempio se un pulsante è premuto aggiornerà il suo

stato, ma non permetterà l'accensione di alcuna luce. Perchè questo sia possibile è necessario introdurre le relazioni tra i Digital Twin, come mostrato nella sezione successiva.

4.4.2 Gestione delle relazioni

Attraverso l'applicazione dell'amministratore è possibile creare ed eliminare le relazioni che intercorrono tra i diversi Digital Twin appartenenti al sistema.

Come già anticipato in precedenza, nel sistema sono possibili due diversi tipi di relazioni:

- la relazione che va da un pulsante ad una luce, identificata dall'id *dtmi:progettotesi:button:control;1*, che indica quale luce è gestita da quale pulsante;
- la relazione che va da pulsante (o luce) ad una stanza. In questo caso esistono due relazioni distinte: una per i pulsanti ed una per le luci, ma il significato della relazione è lo stesso, ovvero l'appartenenza ad una stanza.

Per creare una relazione tra una luce ed un pulsante è necessario selezionare il pulsante e la luce desiderati nelle loro rispettive liste e cliccare su "create relationship". La relazione può essere visualizzata tramite l'Azure DT Explorer o attraverso l'applicazione dell'amministratore (figura 4.7).

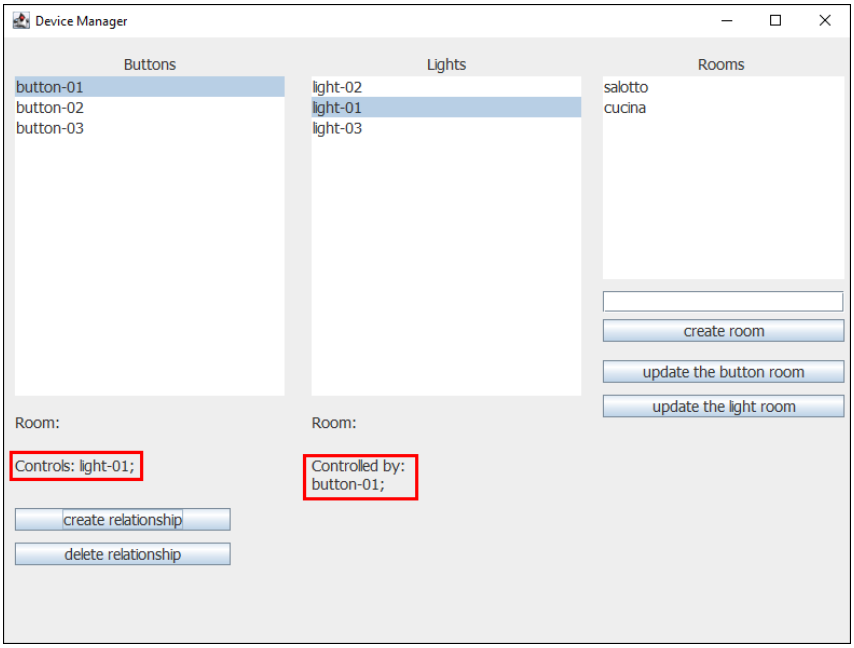


Figura 4.7: visualizzazione della relazione appena creata

Selezionando un pulsante è infatti possibile visualizzare a quali altri Digital Twin esso è legato, ovvero quale luce controlla o a quale stanza appartiene. Lo stesso vale per le luci, con la sola differenza che una luce può essere controllata da più di un pulsante contemporaneamente.

Poiché un pulsante può controllare una sola luce, creando una relazione di tipo *control* su un pulsante che già possiede questa relazione, la vecchia relazione è eliminata in maniera implicita e sostituita dalla nuova.

Così come può essere creata, una relazione può anche essere eliminata: è sufficiente selezionare il pulsante dalla lista e cliccare sul "delete relationship".

Allo stesso modo è possibile creare relazioni tra pulsanti o luci e stanze. In questo caso è sufficiente selezionare il pulsante o la luce dalla rispettiva lista e la stanza desiderata, e cliccare su uno dei due pulsanti sotto la lista *rooms* a seconda di quale DT si vuole legare alla stanza.

L'esempio si conclude inserendo altre relazioni, in modo da connettere tutti i Digital Twin tra loro. In figura 4.8 è mostrato come le relazioni possono essere visualizzate graficamente tramite l'Azure Digital Twin Explorer.

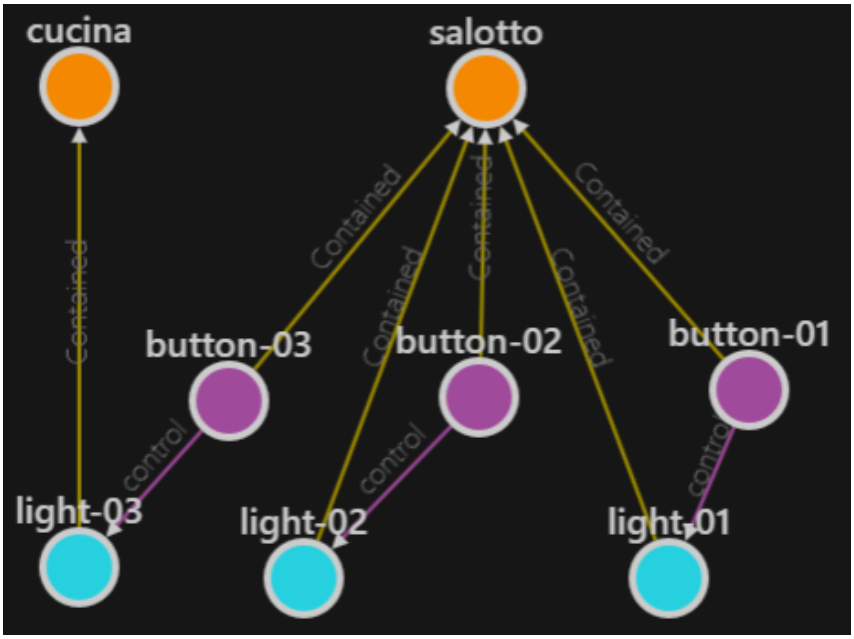


Figura 4.8: visualizzazione delle relazioni attraverso Azure Digital Twin Explorer

Come si può notare dal grafico, perchè un pulsante possa controllare una luce non è necessario che i due si trovino nella stessa stanza. Un pulsante può ad esempio essere nella stanza "salotto" e comandare una luce appartenente alla stanza "cucina".

4.4.3 Shadowing dei dispositivi

L'applicazione descritta precedentemente permette solo operazioni di amministrazione. Per visualizzare lo stato dei DT in tempo reale sono possibili due alternative: la prima è utilizzare lo strumento Azure Digital Twin Explorer, la seconda è creare un'applicazione che si colleghi al Broker MQTT e si iscriva al canale di shadowing in modo da ricevere le notifiche del cambiamento dei Digital Twin. In questo caso per mostrare il funzionamento del sistema sarà utilizzata la prima opzione.

In figura 4.9 è mostrato lo stato del pulsante *button-01* prima e durante il click. Prima della pressione il valore della proprietà *isPressed* è *False*, mentre durante è *True*.

(a) (b)

Figura 4.9: stato del pulsante prima (a) e durante (b) la pressione mostrato attraverso Azure DT Explorer

Come detto in precedenza parlando dell'architettura del sistema, al cambiamento di stato del pulsante, se a questo è collegata una luce, segue l'accensione o lo spegnimento di quest'ultima (figura 4.10). In questo caso il pulsante *button-01* è collegato alla luce *light-01*. Al rilascio del pulsante, sia la proprietà *isPressed* del pulsante, sia la proprietà *isOn* della luce, tornano ad essere *False*, sempre grazie al meccanismo delle relazioni citato in precedenza.

(a) (b)

Figura 4.10: stato della luce *light-01* prima (a) e durante (b) la pressione del pulsante *button-01* mostrato attraverso Azure DT Explorer

4.5 Architetture alternative

Quella descritta nelle pagine precedenti è solo una delle tante architetture possibili che permettono di soddisfare i requisiti esposti nella fase iniziale. Durante la fase di progettazione sono state pensate diverse architetture, ognuna con i propri pro e contro, vediamo ora velocemente quali sono alcune di queste alternative e perchè si è preferito la prima rispetto alle altre.

Relazioni unicamente a livello Azure Digital Twin: La prima alternativa adotta un'architettura pressoché identica a quella descritta nelle pagine precedenti, l'unica cosa che cambia è il modo in cui le relazioni sono memorizzate all'interno del sistema.

Nella soluzione precedentemente descritta, il DT Manager sincronizza le relazioni con quelle dell'istanza di Azure Digital Twin all'avvio dell'applicazione e ne mantiene una copia a livello locale per tutta la durata dell'esecuzione. La soluzione proposta da questa alternativa consiste invece nel non memorizzare le relazioni in locale ma di affidarsi unicamente all'istanza di Azure Digital Twin. In questo caso ogni volta che il pulsante effettua lo shadowing, il DT Manager dovrà interrogare l'istanza di Azure DT per sapere su quale luce eseguire l'azione.

Questa operazione comporta un maggiore delay tra la pressione vera e propria del pulsante e l'accensione della luce, inoltre richiede di effettuare un maggior numero di richieste all'istanza di Azure. Principalmente per questo secondo motivo si è scelto nella soluzione adottata di mantenere una copia locale delle relazioni, in modo da limitare il più possibile il numero di richieste al servizio Azure.

Questa alternativa risulta però estremamente valida e non molto differente da quella descritta nelle pagine precedenti. In entrambi i casi il significato delle relazioni rimane il medesimo, l'unica differenza è come esse sono memorizzate.

Comunicazioni dirette tra Physical Asset: questa seconda alternativa consiste nel far comunicare il pulsante e la luce ad esso collegata attraverso lo stesso topic (figura 4.11). Come nel caso descritto in precedenza, il dispositivo della luce si iscrive al canale *action/device-id*, ma ciò che cambia è chi pubblica messaggi su di esso.

Nell'architettura adottata è il DT Manager a pubblicare i messaggi sul canale *action/device-id*, questo permette di rendere i dispositivi completamente indipendenti l'uno dall'altro. Il pulsante non è a conoscenza dell'esistenza della luce e viceversa la luce non sa dell'esistenza del pulsante.

Perchè il pulsante possa comunicare sullo stesso canale a cui la luce è iscritta è necessario che il broker gli comunichi su quale topic inviare le proprie azioni.

Questo passaggio può essere evitato adottando l'architettura precedentemente discussa.

Inoltre, delegare l'invio delle azioni al DT Manager, consente di limitare le responsabilità dei Physical Asset, che a questo punto si limitano unicamente a eseguire lo shadowing del proprio stato. In una visione più ampia, la stessa classe di pulsanti potrebbe essere utilizzata per diverse applicazioni, ad esempio un primo pulsante potrebbe accendere e spegnere una luce, un secondo alzare una tapparella smart, mentre un terzo abbassare la stessa tapparella. In questo caso, gestendo le relazioni attraverso il DT Manager, il pulsante fisico non dovrà conoscere né il dispositivo a cui è connesso, né tantomeno l'azione da eseguire su di esso o quando inviare tale messaggio.

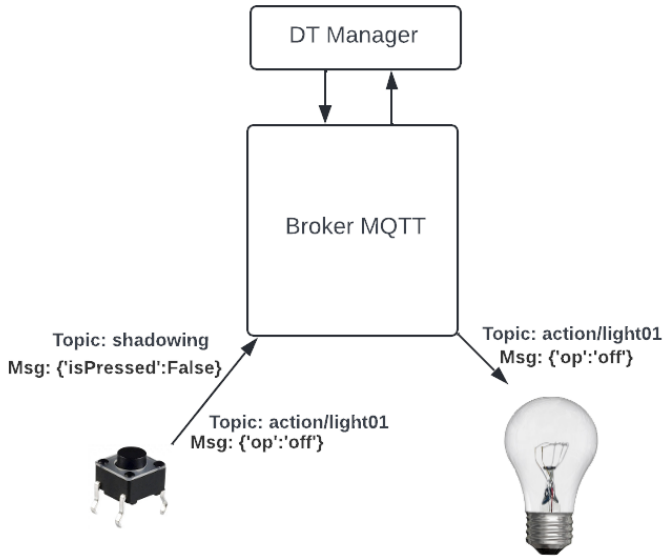


Figura 4.11: Comunicazione tra un pulsante e una luce attraverso lo stesso topic

Connessione diretta PA - Azure Digital Twin: utilizzando le API REST è possibile creare una comunicazione diretta tra i Physical Asset e l'istanza di Azure Digital Twin che contiene i relativi DT. Perchè questa soluzione sia possibile è necessario che i Physical Asset conoscano l'indirizzo relativo all'istanza Azure, o nel caso si voglia introdurre il passaggio intermedio dell'IoT Hub, l'indirizzo relativo all'Hub. Questa soluzione è stata scartata in quanto, nonostante la comunicazione PA-Azure fosse facilmente realizzabile, il problema giunge con la comunicazione a parti inverse, ovvero dall'istanza Azure in direzione Physical Asset.

Inoltre la bassa complessità del sistema, ed il numero limitato di dispositivi che esso contiene, non richiede un'architettura complessa e scalabile, ma è sufficiente un'architettura più semplice come quella proposta. Così facendo inoltre si limitano le dipendenze con l'istanza Azure, limitando quindi i costi di gestione del sistema.

4.6 Considerazioni finali sulla soluzione proposta

L'architettura proposta nelle pagine precedenti soddisfa tutti i principali requisiti del Web of Digital Twin. È possibile creare dinamicamente nuovi Digital Twin e modificare in fase di esecuzione le relazioni che intercorrono tra essi. La struttura proposta permette alle applicazioni esterne di interagire con l'ambiente di Digital Twin attraverso una connessione MQTT, permettendo loro di eseguire sia operazioni di input come le azioni, sia operazioni di output come il tracking dello stato di un Physical Asset.

Nonostante il caso di studio prevedesse la sola gestione di pulsanti e luci, l'architettura scelta risulta essere flessibile ed in grado di gestire qualsiasi tipo di Physical Asset.

Ogni dispositivo, per quanto complesso esso sia, si interfaccia al Broker MQTT attraverso un insieme di topic ben definito. Il canale *createAndBinding* è utilizzato per registrare i nuovi dispositivi connessi ed eventualmente creare i rispettivi Digital Twin, il canale *shadowing* permette ai dispositivi di notificare il proprio cambiamento di stato, mentre il canale *action/dt-id* consente di eseguire azioni sui Physical Asset che lo permettono.

L'architettura proposta consente inoltre una netta divisione delle responsabilità: i dispositivi fisici hanno una bassa responsabilità e si occupano unicamente di comunicare il proprio stato al broker e di reagire alle azioni impartite da esso. Il Broker MQTT si occupa di coordinare le varie componenti, permettendo alle applicazioni esterne di interagire con il sistema, mentre il DT Manager gestisce l'interazione con i Digital Twin veri e propri.

In questa architettura, Azure Digital Twin ricopre un ruolo fondamentale: oltre a mantenere le istanze dei Digital Twin, Azure DT permette di gestire le relazioni in maniera dinamica e grazie al DTDL consente di definire facilmente nuovi modelli attraverso cui creare le istanze. Azure Digital Twin agisce inoltre da layer di persistenza, permettendo di recuperare le informazioni riguardanti il sistema nel caso in cui l'applicazione dovesse interrompere per qualche motivo la propria esecuzione.

Conclusioni

Il lavoro svolto ha mostrato come è possibile realizzare un ecosistema di Digital Twin attraverso la piattaforma Azure Digital Twin, seguendo i principi indicati dal WoDT.

Nonostante il sistema modelli un semplice caso di studio che prevede la gestione di pochi dispositivi, limitati sia in numero che come tipologia, è stato mostrato come in realtà esso permetta di gestire un qualsiasi dispositivo, indipendentemente da quanto complesso quest'ultimo sia.

Nel caso di studio preso in esame, è stato anche mostrato come è possibile gestire dei Digital Twin il cui Physical Asset non è associato ad alcun sistema embedded: è questo il caso della stanza. Prendendo in considerazione contesti applicativi più complessi questa feature si rivela fondamentale, in quanto permette di modellare il comportamento di risorse come processi, attività o luoghi.

In definitiva, Azure Digital Twin si è rivelata essere una piattaforma adatta per la creazione di un sistema Web of Digital Twin: la possibilità di modificare l'ambiente in maniera dinamica, unita alla esistenza delle relazioni, ha permesso di modellare ogni aspetto del WoDT. L'unica differenza tra il modello originale del WoDT e l'architettura adottata per la realizzazione di questo sistema è il modo in cui i dispotivi effettuano lo shadowing. Mentre nella visione originale del WoDT, i Physical Asset svolgono in maniera autosufficiente l'operazione di sincronizzazione, nel sistema proposto il processo di shadowing è invece composto da due passaggi distinti: per prima cosa il dispositivo invia al Device Manager le informazioni riguardanti il proprio stato e sarà poi quest'ultimo, attraverso la componente del DT Manager, ad aggiornare l'istanza del Digital Twin corrispondente. Questa scelta è stata fatta per ridurre la complessità interna dei Physical Asset.

Riguardo il protocollo di comunicazione utilizzato, MQTT si è rivelato essere la scelta ideale in quanto permette di organizzare in maniera ottimale la comunicazione tra i dispositivi, abilitando inoltre funzionalità aggiuntive come quella del tracking.

In tutto questo, il WoDT ha dimostrato come è possibile creare complessi sistemi di Digital Twin, fornendo principi e linee guida da seguire per la loro

realizzazione.

Possibili sviluppi futuri

Come anticipato dall'introduzione, il lavoro svolto si è concentrato maggiormente sulla gestione delle relazioni e nel rendere il sistema dinamico sotto ogni punto di vista. La memorizzazione, al contrario, è stata trattata in maniera teorica ma non è stata implementata all'interno del sistema progettato.

Un possibile sviluppo futuro potrebbe riguardare proprio la gestione della memorizzazione. Come visto nella sezione dedicata, l'ecosistema Azure fornisce tutti gli strumenti necessari perchè la memorizzazione possa essere gestita direttamente al suo interno, anche grazie all'Azure Data Explorer. In alternativa, si può scegliere di affidare la gestione di un semplice database in locale ad una nuova componente all'interno del Device Manager.

Un ulteriore sviluppo futuro potrebbe riguardare il processo di creazione dei Digital Twin. Al momento, quando un dispositivo effettua l'operazione di binding, segnala al Device Manager l'id del modello di cui il Digital Twin deve essere istanza. Per rendere il sistema ancora più dinamico e flessibile si può pensare che in fase di binding il dispositivo indichi direttamente il modello del Digital Twin, così facendo si concede la possibilità, al momento assente, di aggiungere nuovi modelli di Digital Twin in fase di esecuzione.

Ringraziamenti

In primo luogo vorrei ringraziare tutte le persone che mi sono state vicine durante questo percorso durato tre anni: dalla mia famiglia, che ha sempre supportato ogni mia scelta, agli amici, che hanno reso questo percorso meno difficile di quanto realmente non fosse.

Ci tengo inoltre a ringraziare il professor Alessandro Ricci e il dottor Samuele Burattini per avermi seguito durante tutta la stesura della tesi, risolvendo tempestivamente ogni mio dubbio, fornendomi diversi punti di vista e dandomi utili consigli.

Bibliografia

[1] Michael Grieves. Origins of the digital twin concept. *Florida Institute of Technology*, 8, 2016.

[2] Werner Kritzinger, Matthias Karner, Georg Traar, Jan Henjes, and Wilfried Sihn. Digital twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51(11):1016–1022, 2018. 16th IFAC Symposium on Information Control Problems in Manufacturing INCOM 2018.

[3] Ying Liu, Lin Zhang, Yuan Yang, Longfei Zhou, Lei Ren, Fei Wang, Rong Liu, Zhibo Pang, and M. Jamal Deen. A novel cloud-based framework for the elderly healthcare services using digital twin. *IEEE Access*, 7:49088–49101, 2019.

[4] Roberto Minerva and Noël Crespi. Digital twins: Properties, software frameworks, and application scenarios. *IT Professional*, 23(1):51–55, 2021.

[5] Christos Pylianidis, Sjoukje Osinga, and Ioannis N. Athanasiadis. Introducing digital twins to agriculture. *Computers and Electronics in Agriculture*, 184:105942, 2021.

[6] Alessandro Ricci, Angelo Croatti, Stefano Mariani, Sara Montagna, and Marco Picone. Web of digital twins. *ACM Trans. Internet Technol.*, dec 2021. Just Accepted.

[7] Gary White, Anna Zink, Lara Codecá, and Siobhán Clarke. A digital twin smart city for citizen feedback. *Cities*, 110:103064, 2021.

[8] Tetsuya Yokotani and Yuya Sasaki. Comparison with http and mqtt on required network resources for iot. In *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, pages 1–6, 2016.