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 SVILUPPO PROTOTIPALE DI UN MIDDLEWARE PER ECOSISTEMI INTEROPERABILI DI DIGITAL TWINS

Elaborato in

SISTEMI EMBEDDED E INTERNET-OF-THINGS

Relatore

Prof. ALESSANDRO RICCI

Presentata da

ANDREA GIULIANELLI

Corelatore

Prof. ANGELO CROATTI

Prof. MARCO PICONE

Anno Accademico 2020 – 2021

Indice

Introduzione **vii**

1 Digital Twins - Panoramica da ieri ad oggi **1**

1.1 Origini 1

1.2 Ambiti e tecnologie che hanno influenzato maggiormente i Digital Twins 3

1.2.1 Settore manifatturiero 3

1.2.2 Realtà aumentata e virtuale 8

1.2.3 Virtualizzazione 10

1.2.4 Intelligenza artificiale 11

1.3 Impatto nei Digital Twins moderni 12

1.3.1 Caratteristiche 15

1.4 Verso la standardizzazione 29

1.4.1 Digital Twin Consortium 31

1.4.2 National Digital Twin programme 32

1.4.3 Riepilogo 39

2 Proposte architetturali **40**

2.1 DTC Digital Twin System 41

2.2 National Digital Twin 43

2.3 Proposte provenienti dalla ricerca 52

2.4 Riepilogo 57

3 Piattaforme e frameworks disponibili **59**

3.1 Panoramica 59

3.2 Eclipse Ditto 62
3.3 Azure Digital Twins 65
3.4 Riepilogo 74

4 Dai Digital Twins a Web of Digital Twins 76
4.1 Web of Digital Twins 76
4.2 Modellazione in Web of Digital Twins 79
4.3 Considerazioni 81

5 Studio e ideazione di un middleware per Web of Digital Twins 82
5.1 Requisiti del middleware 82
5.2 Proposta architetturale 94

6 Implementazione prototipale 133
6.1 Tecnologie utilizzate 133
6.1.1 Java 134
6.1.2 Vert.x 135
6.1.3 Socket e Web Socket 135
6.1.4 MQTT 136
6.1.5 RDF e RDF Store TDB 137
6.1.6 SPARQL 138
6.1.7 DTDL 139
6.1.8 C# 139
6.1.9 Servizi Microsoft e strumenti di gestione 139
6.2 Implementazione 140
6.2.1 Deploy 140
6.2.2 Implementazione Nodo 147
6.2.3 Implementazione Core 173

7 Valutazione e considerazioni 183
7.1 Piccolo caso di studio 183
7.2 Considerazioni 195

Conclusioni **200**

Ringraziamenti **202**

Introduzione

I *Digital Twins* o "Gemelli Digitali" sono un concetto in notevole crescita che sta diventando protagonista in diversi settori, dal manifatturiero all'healthcare, fino alla gestione intelligente della città.

Essi sono in continua evoluzione ed ancora oggi non si ha una definizione precisa e condivisa in modo standardizzato. Ogni organizzazione sviluppa ed implementa la sua visione. L'aspetto che però ogni visione prevede è che *i Digital Twins si riferiscono all'abilità di clonare un asset fisico attraverso una controparte software. La controparte software (quindi il Digital Twin) riflette tutte le proprietà e le caratteristiche dell'asset fisico importanti per il contesto analizzato* [15].

La definizione dei Digital Twins segue un rapido sviluppo muovendosi da un prodotto industriale ad un concetto molto più generale ed applicabile a praticamente qualsiasi oggetto fisico e non, come ad esempio processi o attività.

In questa tesi verrà svolto uno studio e un approfondimento del paradigma *Digital Twins* prendendo in considerazione la letteratura ad oggi disponibile e le principali piattaforme e tecnologie abilitanti esistenti sul mercato per lo sviluppo di sistemi composti da Digital Twins. L'attenzione verrà poi posta sugli approcci che supportano la realizzazione di ecosistemi di Digital Twins, tra cui *National Digital Twins* e una proposta presente in letteratura chiamata *Web of Digital Twins* [18]. Dopodiché, nella parte progettuale viene presentata una parte incentrata sullo studio e sull'ideazione di un middleware e dell'architettura a supporto per la creazione di ecosistemi interoperabili di Digital Twins. Questo ha portato poi alla realizzazione di un prototipo di tale architettura, validato attraverso un caso di studio. Infatti, l'applicabilità e i

vantaggi di tale soluzione verranno dimostrati attraverso l'applicazione ad un piccolo caso di studio che riguarda il "Major Trauma Management", cioè la gestione del trauma grave.

I capitoli della tesi rappresentano le varie attività che hanno contraddistinto il processo nella costruzione della proposta di architettura. Ho cercato di avere una solida base sullo stato dell'arte e sui principali punti di vista e di sviluppo che caratterizzano, ad oggi, questa tecnologia. Basandomi su di essi e sulla nuova visione offerta in *Web of Digital Twins* ho sviluppato l'architettura e il middleware qui proposti.

Nel primo capitolo è presentata una panoramica del concetto di Digital Twin e delle sue caratteristiche partendo dalla sua nascita fino ai giorni più recenti. Importantissimi i punti di vista di Michael Grieves, di Minerva, Lee e Crespi che hanno contribuito, in modo diverso, a delineare i concetti cardine di questa tecnologia. Inoltre, verranno anche descritti i primi sforzi verso la standardizzazione del concetto e verso la creazione di ecosistemi più complessi.

Nel secondo capitolo vengono presentate le principali proposte architetturali: *Digital Twin System* del Digital Twin Consortium, *National Digital Twin*, e due proposte provenienti dalla ricerca. Queste forniranno ottimi spunti anche per l'architettura sviluppata.

Nel terzo capitolo è presentata una panoramica delle soluzioni attualmente presenti sul mercato con una particolare attenzione ad *Eclipse Ditto* e a *Microsoft Azure Digital Twins* con tutti i servizi ad esso collegati. La soluzione di Microsoft è stata oggetto di un mio particolare studio per tutta la tesi e troverà utilizzo anche all'interno di un componente del progetto.

Nel quarto capitolo viene presentata la visione di ricerca presa come riferimento per lo sviluppo del middleware.

Nel quinto capitolo si entra nel particolare del progetto di tesi fornendo l'analisi dei requisiti del middleware seguita dalla proposta di architettura.

Nel sesto capitolo viene descritta una possibile implementazione dell'architettura e del middleware fornendo prima una panoramica sulle tecnologie utilizzate, poi una descrizione dettagliata dei principali componenti del sistema mostrando le parti di codice più rilevanti.

Nel settimo capitolo viene descritto il piccolo caso di studio su cui il sistema è stato testato e validato e vengono condotte delle considerazioni per eventuali punti di sviluppo futuri.

Capitolo 1

Digital Twins - Panoramica da ieri ad oggi

In questo capitolo si vuole presentare la storia che il concetto di Digital Twin ha vissuto dalla sua creazione fino alle versioni più recenti. Si riporterà una prima definizione data dal suo inventore Michael Grieves, dopodiché basandosi sul survey di Minerva, Lee e Crespi [15] verranno illustrate prima le principali influenze subite dai vari settori e successivamente le caratteristiche più importanti che rappresentano i Digital Twins. Infine, farò una panoramica sui principali sforzi verso la standardizzazione di questo concetto e delle tecnologie associate.

1.1 Origini

Il concetto di Digital Twin fu coniato da Michael Grieves nel 2003 e presentato all'Università del Michigan. Grieves iniziò a dare vita a questo pensiero già nel 2002 durante una presentazione, sempre presso l'Università del Michigan, della sua visione ideale di un sistema PLM[1].

[1]PLM sta per Product Lifecycle Management ed è un'attività di business che riguarda la gestione, nel modo più efficiente possibile, dei prodotti dell'azienda durante tutto il loro ciclo di vita: dalla creazione fino al momento in cui il prodotto viene ritirato dal mercato o smaltito.

La prima definizione data da Michael Grieves fu:

> *"The Digital Twin is a set of virtual information constructs that fully describes a potential or actual physical manufactured product from the micro atomic level to the macro geometrical level. At its optimum, any information that could be obtained from inspecting a physical manufactured product can be obtained from its Digital Twin."*

In questa prima descrizione possiamo notare alcune nozioni che rappresentano le prime versioni del concetto:

- Un Digital Twin si riferisce a un oggetto fisico
- Un Digital Twin contiene le informazioni necessarie per descrivere tutte le caratteristiche di un oggetto fisico oltre al suo comportamento
- Tenendo in considerazione il fatto che il Digital Twin viene applicato durante un processo di PLM, esso prenderà in considerazione tutti i dati dell'oggetto fisico dalla sua ideazione fino alla sua concretizzazione e messa in opera; perciò può descrivere e mettere a disposizione la storia di quell'oggetto fisico.

Grieves ha visto fin da subito l'applicabilità dei Digital Twins nel settore manifatturiero come un concetto e un insieme di tecnologie che potessero affiancare i sistemi di produzione durante tutto il ciclo di vita di un prodotto.

Inizialmente, in [9] Grieves concettualizzò l'idea di un Digital Twin che oggi viene chiamato *descrittivo*, cioè capace di rappresentare lo stato corrente di un oggetto fisico. Questo creava la possibilità di eseguire analisi e osservazioni di prodotti o in generale asset fisici direttamente dalla replica digitale.

Dopodiché, si susseguirono, sempre grazie a Grieves, visioni sempre più complete e ricche di concetti, ad esempio [10] in cui il Digital Twin veniva utilizzato anche per eseguire test e prime forme di simulazione.

Anche altri ricercatori, nel frattempo, iniziavano a dare i primi contributi a questo concetto. In particolare, Minerva, Lee e Crespi in [15] individuarono i

settori e le tecnologie che influenzarono maggiormente il concetto e sopratutto un *insieme di caratteristiche* che, ad *oggi*, vengono considerate la *base del concetto di Digital Twin.*

1.2 Ambiti e tecnologie che hanno influenzato maggiormente i Digital Twins

A partire da ciò che Grieves definì nel 2003, i ricercatori di tutto il mondo si mobilitarono per dare un proprio contributo al concetto al fine di espanderlo, completarlo e renderlo qualcosa di più maturo, con potenzialità che inizialmente non erano state esplorate.

A seguito delle ricerche, i Digital Twins furono integrati con l'IoT, l'IIoT, l'intelligenza artificiale, l'apprendimento automatico e le tecniche di analisi dei dati. Inoltre, furono creati dei modelli di simulazione che aggiornandosi in modo sincrono con le loro controparti fisiche riuscivano a fornire "what-if scenario". Queste tecnologie vengono sfruttate in modo diverso a seconda del campo di utilizzo e sono tutte tecniche che venivano o vengono utilizzate in altri settori e a cui molti ricercatori e molte aziende si sono ispirati per definire la loro visione di Digital Twin. Per questo motivo è utile avere una rapida panoramica delle maggiori influenze che il concetto di Digital Twin ha subito dai vari settori e dalle tecnologie [15].

1.2.1 Settore manifatturiero

Il primo settore a muoversi nella direzione dei Digital Twins, quindi quello che ha dato il maggior contributo iniziale, è stato il manifatturiero. Infatti, Grieves pone proprio le origini del termine nel contesto manifatturiero grazie ai sistemi PLM, capaci di gestire il ciclo di vita di un prodotto [9][10].

Il settore manifatturiero, oltre ad influenzare i Digital Twins nella definizione di molti aspetti e caratteristiche, ha rappresentato anche uno dei primi campi in cui sviluppare, verificare e testare la reale applicabilità e i vantaggi

dei Digital Twins in un contesto su larga scala e ad elevata complessità. Ha portato con sé numerosi approcci che poi hanno influenzato anche altre aziende o anche gli stessi ricercatori nella progettazione e nello sviluppo delle loro soluzioni per l'utilizzo dei Digital Twins in generale, anche fuori da questo settore.

Prima dei recenti sistemi IoT, l'unico modo per avere una conoscenza dello stato di un oggetto fisico, di un macchinario o dell'azienda in generale era quello di essere in sua prossimità per poter effettuare delle analisi. Quindi, le informazioni riguardanti l'oggetto fisico erano praticamente inseparabili dallo stesso oggetto fisico. Solamente grazie alle tecnologie recenti è stato possibile estrapolare e rendere disponibili le informazioni di un asset fisico esternamente ad esso e creare quello che oggi chiamiamo Digital Twin.
Le informazioni, di qualsiasi natura esse siano, sono di fondamentale importanza e questo viene sottolineato in [10] in cui Grieves definisce i *"Sistemi Complessi"*. I sistemi complessi (*Complex Systems*) sono caratterizzati da una grande rete di componenti con comunicazioni molti-a-molti e con un'elaborazione delle informazioni complessa che rende la predizione dello stato di questi sistemi molto complicata.
I comportamenti di questi sistemi possono essere rappresentati in quattro categorie:

- *Predetti e desiderati (PD)*: sono i requisiti che il nostro sistema deve rispettare, sono decisi in fase di creazione e design del prodotto

- *Predetti e non desiderati (PU)*: sono i comportamenti problematici, ma che sono stati già presi in considerazione e verranno eliminati nelle fasi successive

- *Non predetti e desiderati (UP)*: sono comportamenti che suscitano sorpresa, però senza causare problemi, anzi sono vantaggiosi

- *Non predetti e non desiderati (UU)*: questi comportamenti rappresentano la sorgente di potenziali problemi seri del sistema.

Un Digital Twin, in questo contesto, deve essere in grado di rappresentare ed agire come un oggetto reale durante tutto il suo ciclo di vita, permettendo il design, la prototipazione e il testing su una rappresentazione virtuale di un asset fisico. Infatti come detto in [15] e [10], il Digital Twin può essere utilizzato al fine di determinare tutti i comportamenti, soprattutto gli UU, nelle fasi iniziali del ciclo di vita di un prodotto, addirittura quando questo ancora non esiste.
Sempre più spesso il Digital Twin viene creato direttamente nella fase di design, ancora prima che il prodotto esista. In questo modo è possibile eseguire test e simulazioni sul Digital Twin al fine di scovare i problemi.

Il fatto di avere un prodotto ancora prima della sua reale esistenza è un concetto molto forte. Basti pensare che prima di quelli che oggi chiamiamo "spazi virtuali", le persone potevano solamente immaginarsi il prodotto e "testarlo" nella loro mente, grazie al puro ragionamento. Oggi, invece, questi spazi virtuali in congiunzione con concetti abilitanti come quello dei Digital Twins aprono la via ad una rivoluzione intersettoriale.

Ruolo dei Digital Twin nelle varie fasi del ciclo di vita di un prodotto

Come detto precedentemente ed illustrato nella Figura 1.1, i Digital Twins trovano impiego in tutte le fasi del ciclo di vita di un prodotto dove, grazie alla controparte software, aiutano nel migliorare ed ottimizzare il prodotto stesso.

Durante la fase di Design, il Digital Twin può essere utilizzato per scegliere le migliori opzioni e caratteristiche del prodotto o ad esempio per simulare prestazioni. Durante la fase di Produzione invece, viene utilizzato principalmente per la simulazione dei PD (comportamenti predetti e desiderati), cioè quelli richiesti dai requisiti del prodotto stesso, al fine di verificare la conformità. Mentre, durante la fase operativa del prodotto può essere utilizzato per controllare e prevedere malfunzionamenti prima che questi avvengano, oppure può essere testato in situazioni limite che potrebbero verificarsi in futuro.

Potendo eseguire le simulazioni direttamente sul Digital Twin all'interno

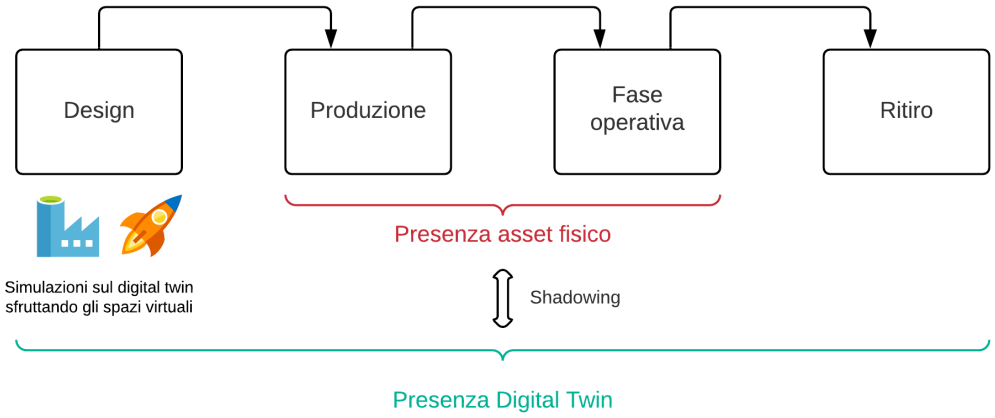

Figura 1.1: Ciclo di vita di un asset fisico e del Digital Twin associato nel sistema manifatturiero

di uno spazio di simulazione virtuale si evita di testare le caratteristiche e i limiti del prodotto direttamente su prototipi i quali hanno un elevato costo di progettazione. Questo ci consente di risparmiare somme di denaro notevoli in quanto possiamo testare ogni aspetto virtualmente ed inoltre ci consente di effettuare anche molte più simulazioni, cercando di prevedere il comportamento dell'asset fisico in situazioni sempre più particolari e in ambienti diversi permettendo di scovare molti più UU (comportamenti non predetti e non desiderati) di quanto era possibile fisicamente. La simulazione e la predizione sono utili anche nelle fasi operative del prodotto. Infatti, se abbiamo le caratteristiche dell'ambiente in cui il prodotto è immerso è possibile prevedere il suo comportamento e, in caso di una situazione di pericolo, il Digital Twin, sfruttando una comunicazione bidirezionale con l'asset fisico, potrà comandare un cambiamento di stato (prendendo una vera e propria decisione; più avanti si parlerà in questo contesto di Cognitive Digital Twin).

Quindi vediamo come l'uso di informazioni può ridurre di molto lo spreco di risorse fisiche [10], il quale è un aspetto molto importante alla luce di tutti i problemi ambientali che caratterizzano il mondo di oggi.

Si nota che le tecnologie offerte dall'IoT sono qui di fondamentale importan-

za per assicurare il link (o shadowing come illustrerò più avanti) tra il Digital Twin e l'asset fisico rappresentato. Esse ci consentono di utilizzare i sensori presenti sull'asset fisico per poter mantenere la controparte software virtuale (il Digital Twin) aggiornata e allo stesso tempo di eseguire operazioni ed azioni su di essa come se le stessimo eseguendo sull'asset in questione.
Inoltre, di fondamentale importanza sono le tecniche di Data Science e di Intelligenza Artificiale che vengono utilizzate al fine di effettuare previsioni e simulazioni. Quindi, una parte centrale è anche la raccolta dei dati provenienti dagli asset fisici in modo da tracciare la storia dei Digital Twins e poter utilizzare quei dati per analisi e per migliorare i modelli di machine learning esistenti.

Valutazione di un Digital Twin all'interno di un sistema manifatturiero

Grieves, sempre all'interno dell'ambito manifatturiero, propose un test di valutazione per i Digital Twins simile a quello proposto da Alan Turing per i sistemi di intelligenza artificiale *"Imitation Game"* [2].

Grieves elaborò una serie di test chiamati *"Grieves' Tests of Virtuality (GTV)"* al fine di valutare la bontà di un sistema virtuale di rispecchiare la propria controparte fisica [10].
La premessa di GTV è molto simile a quella dell'Imitation Game di Turing. Un osservatore umano viene posizionato in una stanza con due schermi. Uno schermo visualizza un video (eventualmente in real-time) di una stanza reale in cui un sistema fisico è presente. L'altro schermo è connesso ad un computer. Ci sono tre test che l'osservatore può compiere:

- Sensory Visual Test: l'osservatore può richiedere qualsiasi manipolazione dell'oggetto nello spazio: rotazioni, sollevamenti o addirittura disassem-

[2]La premessa del test di Alan Turign era che un osservatore, isolato da una persona e da un computer, potesse fare domande ad entrambi. Se l'osservatore non fosse stato in grado di capire quando rispondeva il computer e quando la persona, allora il computer avrebbe passato il test.

blamenti. Se l'osservatore non riesce a distinguere il sistema fisico da quello virtuale allora il Sensory Visual Test è superato.

- Performance Test: l'osservatore può richiedere l'esecuzione di qualsiasi azione, ad esempio: applicare una forza, sottoporlo a stress o inserirlo all'interno di una galleria del vento. Se l'osservatore non riesce a distinguere il comportamento del sistema fisico da quello virtuale allora il Performance Test è superato.

- Reflectivity Test: l'osservatore può richiedere lo stato di ogni aspetto del prodotto. Se l'osservatore riesce ad ottenere le stesse informazioni (in quantità e qualità) che si otterrebbero dal sistema fisico, dal sistema virtuale allora il Reflectivity Test è superato.

Oggigiorno abbiamo un quantità tale di tecnologie che il superamento dei tre test è sempre meno costoso e dispendioso. Infatti, per quanto riguarda le tecnologie sensoriali, soprattutto visive, si riescono ad elaborare modelli 3D che rispecchiano in modo realistico il prodotto fisico reale. Dal punto di vista del Reflectivity Test, le nuove tecnologie offerte dall'IoT ci consentono di mantenere un link con il prodotto anche dopo che ha lasciato la fabbrica grazie alla possibilità di inserire microprocessori in praticamente qualsiasi prodotto. L'unico test che richiede ancora molte risorse per essere superato a pieno è il Performance Test in quanto è necessario che lo spazio virtuale in cui il sistema virtuale è inserito abbia tutte le leggi fisiche dell'universo codificate al suo interno. Invece, sempre più spesso ci si limita a creare spazi virtuali con "solo" le leggi fisiche strettamente legate al contesto di inserimento del sistema, in modo da poter effettuare simulazioni in linea con il suo reale utilizzo.

1.2.2 Realtà aumentata e virtuale

Altri grandi protagonisti dell'ultimo decennio sono la realtà virtuale e la realtà aumentata le quali hanno applicazioni che spaziano dal gaming ai film, passando per la medicina e sono tutt'ora in espansione. La realtà virtuale è la tecnologia più ampiamente nota. È completamente immersiva creando un

ambiente interamente virtuale dove l'utente può interagire con oggetti logici (virtuali) i quali possono avere anche una controparte fisica reale. La realtà aumentata invece, sovrappone le informazioni digitali agli elementi del mondo reale e quindi va ad *aumentare* la quantità di informazioni associate all'oggetto fisico in modo da rappresentare l'oggetto e la sua forma aumentata come se fossero un'unica entità. Quindi l'oggetto fisico viene migliorato rendendolo più interessante e ricco di significato per l'utente.

Le tecniche di realtà aumentata e virtuale sono state sviluppate al fine di simulare, descrivere e costruire ambienti virtuali che possano essere legati a quelli fisici per diversi scopi: unire elementi distanti geograficamente, aumentare le funzionalità di un oggetto fisico, creare un'esperienza più immersiva, assistere persone con problematiche, interagire in ambienti in cui è possibile visualizzare elementi a loro volta virtuali ecc..

In [15] si evidenzia una stretta correlazione tra il concetto di Digital Twin e l'Avatar della realtà virtuale. La realtà virtuale infatti, è caratterizzata dalla presenza di Avatar i quali sono rappresentazioni virtuali, inserite all'interno di un ambiente anch'esso virtuale, di un oggetto, come una persona, che agisce rispecchiando la controparte fisica. L'Avatar si prende carico di azioni al fine di raggiungere gli obiettivi, raccoglie dati e interagisce con altri elementi all'interno dell'ambiente virtuale a nome della controparte fisica.

Per poter rappresentare gli oggetti e gli eventi dell'ambiente reale è necessario avere un modello che rappresenti nel modo più fedele possibile le varie controparti fisiche. A tal fine, possono essere utilizzate semantiche e ontologie[3] in modo da rappresentare l'ambiente e poter ragionare su di esso in maniera più standardizzata. Infatti, rappresentando un modello attraverso un'ontologia condivisa, lo stesso modello può essere condiviso con i sistemi che utilizzano la stessa ontologia al fine di estrarre insights maggiori.

Inoltre, si possono estrapolare alcuni concetti dalle architetture software che alimentano i sistemi di realtà virtuale che sono di notevole importanza per

[3]Un'ontologia è una rappresentazione formale, condivisa ed esplicita di una concettualizzazione di un dominio di interesse. Un esempio di ontologia è FHIR di HL7.
`https://www.hl7.org/fhir/`

i Digital Twins:

- Un flusso continuo di dati ed eventi mantiene la rappresentazione virtuale sincronizzata con la controparte fisica (shadowing). Infatti, l'asset fisico subisce dei cambiamenti durante il suo ciclo di vita: stato, eventi, comportamenti ecc.. Queste trasformazioni devono essere rispecchiate all'interno della rappresentazione virtuale.

- Track continuo dello spazio e del tempo in modo tale da creare la storia dell'asset. I dati storici dell'asset, una volta memorizzati a lungo termine, possono essere impiegati per diversi scopi: analisi, previsioni, simulazioni.

- Necessità di avere un modello che rappresenti lo stato e il comportamento dell'asset fisico con un livello di astrazione che dipende eventualmente dal contesto.

- Allineare lo spazio virtuale rispetto ai vincoli del mondo fisico.

Quindi si può notare come l'influenza dei sistemi di realtà virtuale e aumentata portino notevoli concetti all'interno del mondo dei Digital Twins. Infatti, evidenziano l'eventuale necessità di scambi di dati in real-time al fine di consentire la sincronizzazione con gli asset fisici e sopratutto la possibilità di rappresentare e modellare oggetti della realtà via software.

Inoltre, questi sistemi rappresentano degli importanti strumenti per aumentare la user experience nei progetti che coinvolgono Digital Twins, fornendo rappresentazioni grafiche molto complete ed immersive.

1.2.3 Virtualizzazione

La virtualizzazione, cioè "l'insieme di tecniche che consente la creazione di un ambiente di elaborazione simulato, o virtuale, rispetto a un ambiente fisico", ha anch'essa contribuito alla definizione di alcuni punti che rappresentano il concetto di Digital Twin. Innanzitutto, partendo dalla definizione data possiamo notare che *ci consente di creare un ambiente di elaborazione virtuale ... rispetto a un ambiente fisico*, in linea perfetta con l'idea di replica virtuale;

ma oltre a questo, essa è il punto di partenza per la realizzazione di servizi che vedono i Digital Twins come elemento centrale.

Infatti, i Digital Twins ci consentono di creare una rappresentazione virtuale di un asset fisico che si trova all'edge della rete pubblica, eseguirlo all'interno del proprio framework il quale si trova, ad esempio, a livello di cloud, e grazie alla virtualizzazione fornire servizi in modo scalabile a praticamente tutto il mondo, sollevando quindi l'asset reale dall'onere di rispondere a milioni di richieste (Sezione 1.3 - Replication).

Un ulteriore contributo viene fornito anche a livello concettuale per la creazione di una possibile architettura di gestione dei Digital Twins. L'architettura dei Digital Twins deve gestire, oltre alle rappresentazioni e i modelli virtuali, anche il link continuo con l'asset fisico, oltre che all'API per poter comunicare con i Digital Twins; perciò, saranno necessari meccanismi tipici dei sistemi di virtualizzazione come: gestione dei fault, configurazioni, analisi delle performance, scalabilità e sicurezza.

1.2.4 Intelligenza artificiale

L'ultimo aspetto considerato di influenza per il concetto di Digital Twin è dato dall'intelligenza artificiale e dai sistemi multi-agente. L'intelligenza artificiale fornisce un insieme di tecnologie in grado di estrapolare una conoscenza e una comprensione di un insieme di fenomeni e in alcuni casi decidere le azioni da compiere.
I sistemi multi-agente si basano su un insieme di agenti che agiscono all'interno di un ambiente interagendo tra loro e raccogliendo dati al fine di raggiungere un determinato obiettivo.

I sistemi multi-agente condividono con i Digital Twins una serie di aspetti:

- Promuovono una collaborazione estesa tra agenti attraverso lo scambio di informazioni e dati supportati da specifici protocolli o API. Infatti in una visione più estesa dei Digital Twins, si potrebbe pensare che questi collaborino tra di loro al fine di raggiungere un obiettivo comune o più semplicemente per estrarre insights di maggior valore e qualità.

- Gli agenti collezionano una grande quantità di dati al fine di migliorare la propria conoscenza dell'ambiente in cui operano allo stesso modo dei Digital Twins al fine di implementare la storia dell'asset e poter utilizzare i dati per modelli di ML o AI.

- È necessario che ogni agente sia identificato univocamente in modo da potervi accedere in qualsiasi momento.

- Pongono l'attenzione sulla simulazione, permettendo di simulare il loro comportamento in specifici ambienti con particolari caratteristiche.

Inoltre, i sistemi multi-agente introducono anche il concetto di *Cognitive Digital Twin* [15][18] partendo dal presupposto che un "Cognitive Agent" è un particolare tipo di agente che ha l'abilità di applicare operazioni computazionali costose e cognitive come ad esempio prendere decisioni ed agire in base ad esse. Essi, fanno grande affidamento alle recenti tecniche e scoperte in Intelligenza Artificiale.

1.3 Impatto nei Digital Twins moderni

Grazie all'influenza di questi quattro elementi: settore manifatturiero, realtà aumentata e virtuale, virtualizzazione e sistemi multi-agente, il concetto di Digital Twin è stato espanso e arricchito da nuove idee, pensieri e visioni. Ogni elemento ha, attraverso la sua visione, delineato l'idea di creare un'entità software che può rappresentare completamente, rispecchiare e soprattutto estendere il comportamento, lo stato e le funzionalità di un asset fisico [15].

Inoltre, hanno spianato l'idea di un Digital Twin che non si limita a rispecchiare fedelmente lo stato del suo asset fisico, ma che può, a sua volta, azionare e comandare la sua controparte fisica, andando a creare un ciclo "infinito" in cui il Digital Twin raccoglie dati dall'asset, li elabora ed invia informazioni (comandi) all'asset a seconda dello stato corrente.

Quindi questa visione di Digital Twin è composta da tre dimensioni come riportato in [18]: asset fisico, rappresentazione virtuale e parti di comunicazio-

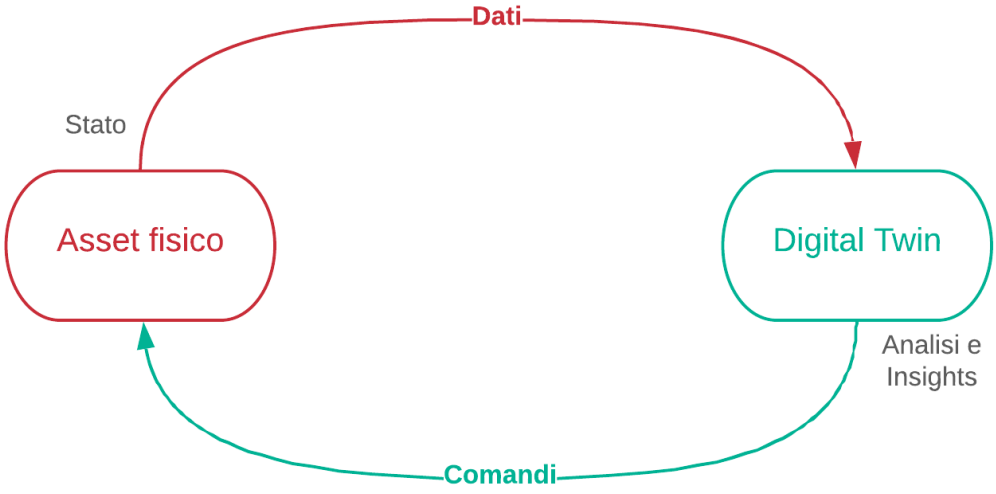

Figura 1.2: Ciclo dei dati in un Digital Twin

ne. La parte virtuale rappresenta il Digital Twin o meglio la rappresentazione software (replica) dell'asset fisico ed è mappata nello spazio reale attraverso le parti di comunicazione che consentono lo scambio di informazioni.

Già nella definizione originale [10] la struttura del Digital Twin veniva rappresentata attraverso un modello che includesse, oltre allo stato, funzionalità e comportamento.

Inoltre, il processo che vede protagonista il link con l'asset fisico (quindi la linea rossa nella Figura 1.2) è uno dei principali del concetto di Digital Twin ed è lo *shadowing*. Lo shadowing è il processo che consente l'aggiornamento dello stato interno di un Digital Twin in real-time o near-real-time con i dati provenienti dal sistema fisico e raccolti da diverse fonti: sensori, piattaforma IoT, ERP[4], PLM[5], ecc.. e trasferiti digitalmente al Digital Twin.

L'unico problema è che le caratteristiche descritte fin'ora sono molto limitanti in termini di dominio applicativo, in quanto sono molto legate al dominio del problema affrontato.

È necessario uno sforzo per vedere queste caratteristiche ad un livello più

[4]Enterprise resource planning
[5]Product Lifecycle Management

alto, astratto e allo stesso tempo delineare proprietà e funzioni che i Digital Twins devono possedere, andando a creare una definizione di essi che sia indipendente dal dominio applicativo.

Minerva, Lee e Crespi in [15] propongono una definizione e un insieme di proprietà che i Digital Twins devono possedere per essere chiamati tali. Possiamo vedere queste proprietà come un primo passo verso la visione moderna dei Digital Twins a cui si ispira anche la visione nuova e più aperta presentata in questa tesi successivamente.
Riporto la definizione:

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

Si può notare un incredibile passo avanti dato dal lavoro dei ricercatori. Infatti, inizialmente si parlava solo di rispecchiare la controparte fisica, qui invece, parla di rispecchiare proprietà, condizioni e comportamento attraverso *modelli* e dati. Inoltre, si parla, come abbiamo descritto precedentemente, del concetto di simulazione e del fatto che il Digital Twin copre la sua controparte fisica per tutto il ciclo di vita.

Quindi, questa definizione rappresenta il punto d'incontro di tutte le influenze che fino a quel punto il concetto di Digital Twin ha subito dai quattro elementi principali che ho descritto. Però, si rimane ancora molto legati al fatto che un asset fisico sia materiale in natura e non un qualcosa che può anche non avere una forma fisica nella realtà, ma esistere ugualmente; questo sarà un punto di svolta nella nuova visione presentata in questa tesi nel Capitolo 4.

1.3.1 Caratteristiche

Di seguito andrò a descrivere le proprietà che i moderni Digital Twins devono possedere per poter essere chiamati tali in linea con la visione proposta in [15] e da cui si prenderà spunto per la definizione in [18].

Identità

Oggigiorno viviamo in una società in cui la gestione della complessità è di fondamentale importanza. Ogni persona, ogni prodotto, ogni elemento della società deve poter essere univocamente identificabile in modo da poter essere individuato e tracciato all'interno dell'ambiente o del contesto in cui vive o risiede. Inoltre, sempre più importante, sopratutto in questo periodo di pandemia da Covid-19, é l'identificazione nello spazio e nel tempo. In questo modo, oltre ad essere individuati come tali, possiamo capire anche la posizione all'interno della storia, del tempo e del mondo di un particolare elemento appartenente allo spazio reale. Senza il tracciamento nello spazio tempo, la gestione della pandemia sarebbe stata molto più complicata e sopratutto molto meno efficiente. Quindi, sopratutto in questi periodi si rileva di fondamentale importanza un'attenta gestione dell'identità.

Tenendo in considerazione la definizione di Digital Twin in particolare: *"A Digital Twin is a comprehensive software representation of an individual physical object .."*, si evidenzia la necessità di avere una forma di identificazione anche per le repliche digitali la quale sia allo stesso livello di complessità, se non più elevato, delle controparti fisiche. Perciò, al fine di poter essere individuate all'interno dello spazio software, le repliche digitali dovranno avere un proprio identificatore univoco che prenda in considerazione anche lo spazio e il tempo.

Con queste considerazioni è sempre possibile utilizzare la replica digitale al fine di rappresentare la controparte fisica nello spazio e nel tempo. Si pensi ad esempio al Digital Twin di un paziente, attraverso l'identificazione del paziente riusciamo a capire di quale persona si tratta, mentre attraverso l'identificazione nello spazio e sopratutto nel tempo è possibile stabilire lo stato delle

informazioni e del modello (quindi l'esatta replica) rappresentando il paziente nel contesto necessario per l'analisi (Figura 1.3).

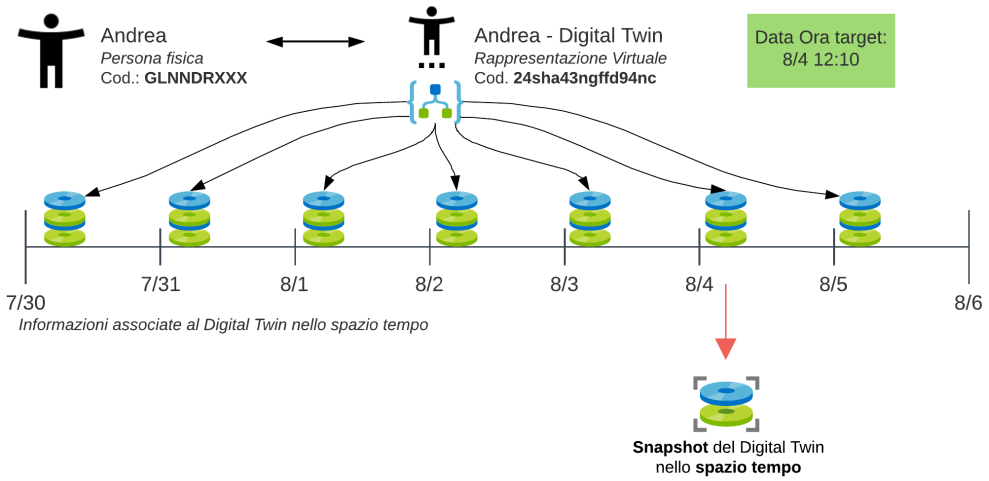

Figura 1.3: Individuazione di un Digital Twin e ottenimento dello stato nello spazio tempo

Ovviamente, più repliche potranno far riferimento allo stesso asset fisico. In tal caso, la replica digitale è comunque individuata tramite l'identificatore del Digital Twin e il riferimento all'identificatore dell'asset fisico.

Rappresentatività e contestualizzazione

Un asset fisico è descritto da attributi, proprietà e comportamenti. Tutti questi elementi che rappresentano l'asset fisico potrebbero non essere di fondamentale importanza in tutti i contesti di analisi e quindi in alcuni casi le informazioni necessarie per raggiungere gli obiettivi potrebbero essere meno oppure derivate da quelle elementari.

Allo stesso tempo, il Digital Twin di un asset fisico deve essere il più possibile verosimile all'originale. Tuttavia, modellare un oggetto fisico in tutte le sue sfaccettature è spesso difficile e a volte un task che non porta vantaggi a livello applicativo. Infatti, come sottolineato, è sempre possibile determinare

da un progetto una serie di scopi e obiettivi rientranti in un contesto target nel quale si opera.

Per questo motivo, il Digital Twin dovrebbe essere descritto da un modello che viene progettato ed implementato al fine di rappresentare tutte quelle proprietà, caratteristiche e comportamenti che sono necessari e sufficienti per qualificare quella rappresentazione virtuale come rappresentativa dell'asset fisico sotto tutti i punti di vista e sotto tutte le funzionalità rientranti nel contesto target.

La *Rappresentatività* di un Digital Twin dipende dalla similarità rispetto all'asset fisico a cui vogliamo arrivare (quindi la qualità di riproduzione delle caratteristiche della controparte fisica) e dalla contestualizzazione.

Contestualizzare un Digital Twin significa che tutte le caratteristiche e i dati del modello sono necessari e sufficienti per rappresentare l'asset fisico nello specifico contesto in considerazione.

Riflessione

La Riflessione è una proprietà che completa quelle di Rappresentatività e di Contestualizzazione. Essa punta l'accento sul fatto che il modello rifletta correttamente gli attributi, le caratteristiche, lo stato, gli eventi e le azioni su cui è necessario concentrarsi. È importante capire se il modello è abbastanza rappresentativo e se questo consenta di riflettere in modo corretto l'asset fisico. Inoltre, la riflessione riguarda anche la capacità dell'asset fisico di poter essere misurato e rappresentato nel rispetto degli obiettivi applicativi.

Ogni valore rilevante della controparte fisica deve essere *univocamente* rappresentato nel Digital Twin. Ad ogni valore può essere applicata una funzione di trasformazione $f(x)$ con la quale mappare i dati. La funzione $f(x)$, dove x sono i dati dell'asset fisico e f è la trasformazione dei dati all'interno della replica, può essere semplicemente una funzione identità (e quindi salvo i dati invariati) oppure potrebbe essere una vera e propria funzione che mi consente anche di ottenere dati derivati. Essa non deve obbligatoriamente essere

iniettiva; con le nuove tecniche di AI, valori differenti possono contribuire a determinare lo stesso valore sulla replica logica.

Replication

Come detto, la virtualizzazione ha dato grandi contributi alla definizione del concetto di Digital Twin offrendo un meccanismo per la creazione di ambienti di elaborazione virtuali in cui creare repliche di asset fisici.
Con *Replication* si intende proprio questa capacità di replicare un asset fisico in diversi ambienti. Al fine della replicazione, l'asset fisico viene appunto virtualizzato e replicato in molteplicità all'interno di uno spazio virtuale.
È possibile individuare diversi pattern di virtualizzazione di un asset fisico in un Digital Twin. Il più semplice è quello che vede una mappatura 1-1 tra l'asset e la replica. In questo modo il Digital Twin è come se fosse la copia esatta dell'asset all'interno del mondo virtuale.
Altri pattern comuni invece mirano a sfruttare le potenzialità computazionali dei servizi cloud i quali fanno largo uso della virtualizzazione.

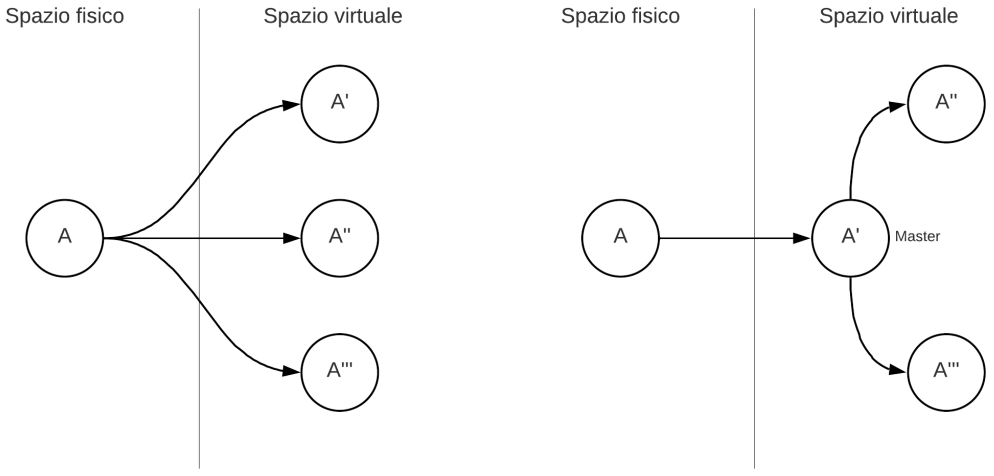

Figura 1.4: Pattern comuni di replicazione

In Figura 1.4 si nota proprio questa caratteristica di alcuni pattern.

Essi ci permettono di rendere disponibili servizi presenti agli edge della rete pubblica ipoteticamente a tutto il mondo, sfruttando le piattaforme cloud e la virtualizzazione. In questo caso un asset *A* viene virtualizzato in due modi. Nel primo (Figura 1.4 a sinistra) vediamo la creazione di tre repliche a partire dall'asset A che sono direttamente collegate all'asset stesso e quindi in sincronizzazione diretta con la controparte fisica. Questo approccio può presentare diversi problemi di scalabilità. Infatti, se l'asset A fosse un semplice dispositivo agli edge della rete (come spesso accade) allora non avrebbe abbastanza risorse computazionali per poter far fronte a migliaia o milioni di richieste simultanee o molto ravvicinate tra loro in quanto essendo un dispositivo fisico non può replicarsi a sua volta o scalare per poter rispondere a multiple richieste di polling.
Per questo motivo, negli scenari ad elevato traffico si adotta la strategia a destra della Figura 1.4 in cui vengono sempre fatte le tre (il numero è solo riferito all'esempio) repliche virtuali nel quale solo una di esse è collegata e in sincronizzazione diretta con l'asset fisico; essa prende il nome di *Master Replica*. Dopodiché le altre repliche si sincronizzeranno solo con la Master Replica andando a liberare l'asset fisico da tutte le richieste di polling visto che ora riceverà solamente quelle del Master. Quindi, solo le repliche faranno fronte alle innumerevoli richieste del livello applicativo e avremo la possibilità di creare quante repliche vogliamo, seguendo sempre lo stesso pattern (magari anche a più livelli, in cui è presente un layer di master che alimenta un ulteriore layer di repliche), assicurando scalabilità e continuità di servizio grazie alle tecniche di gestione dei fault presenti nei sistemi di Virtualizzazione (Figura 1.5).

Ogni replica comunque mantiene lo stato dell'asset fisico e a livello di framework si deve assicurare che lo stato tra le varie repliche sia sempre consistente al fine di erogare il servizio nella maniera corretta con la stessa visione condivisa della realtà.

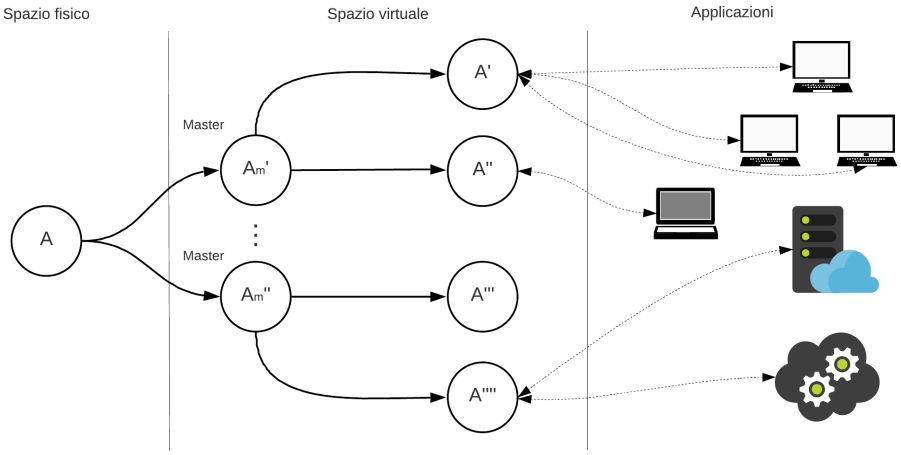

Figura 1.5: Possibile pattern di Replication a più livelli con interazione del layer Applicativo

Entanglement

Il Digital Twin deve sempre avere una visione aggiornata dello stato corrente dell'asset fisico. A tal fine è necessario un collegamento (un link) tra l'asset fisico e la sua replica virtuale che assicuri il trasferimento dei dati e delle informazioni al Digital Twin in real-time o near-real-time. In questo modo, il Digital Twin potrà mettere a disposizione queste informazioni alle applicazioni e ai servizi che intendono utilizzarli.

L'asset fisico può trovarsi in qualsiasi parte del mondo e avere vincoli di disponibilità o di comunicazione, perciò a seconda del contesto e degli obiettivi da raggiungere (anche in termini di performance) è necessario stabilire i requisiti del link tra asset e Digital Twin.

Solitamente si considerano tre caratteristiche fondamentali:

- Connettività: la comunicazione a seconda della disponibilità in termini computazionali e di rete può essere diretta o indiretta.
  La comunicazione diretta prevede che l'asset fisico e il Digital Twin siano in comunicazione senza alcun intermediario a consentire lo scambio di

dati. Ovviamente non viene considerato come intermediario il framework o la piattaforma su cui i Digital Twins vengono memorizzati e gestiti. La comunicazione indiretta si presenta invece nel momento in cui l'asset fisico per vari motivi legati alla posizione, alle proprie capacità o a svariati vincoli non è in grado di comunicare in prima persona i propri dati, perciò deve fare affidamento su una terza parte che può agire da gateway oppure da osservatore. Nel primo caso l'asset invia i dati al gateway utilizzando tecniche o protocolli di comunicazione diversificati (ad esempio Zigbee) il quale poi si occupa di effettuare la sincronizzazione con il Digital Twin. Nel secondo caso invece l'asset è proprio un elemento quasi statico, il quale non è in grado di memorizzare, processare, né tanto meno inviare dati, perciò l'unico modo per consentire una raccolta dati è quello di utilizzare una terza parte che osserva (attraverso telecamere, sensori o tramite l'ambiente in cui è inserito) l'asset ed invia i dati elaborati alla piattaforma che gestisce il Digital Twin.

- Prontezza: ogni tipologia di asset o di dominio applicativo presenta requisiti temporali diversi. Il Digital Twin deve poter rispecchiare lo stato attuale della propria controparte fisica. Per poter far ciò occorre che l'asset fisico sia in grado di sostenere uno scambio di informazioni che permetta alla replica virtuale di non perdere alcun evento. In particolare, è necessario che il tempo di aggiornamento del Digital Twin sia inferiore al tempo medio di cambiamento di stato dell'asset fisico.
  Per alcuni tipi di domini è richiesto che l'asset o la terza parte che si occupa della sincronizzazione abbia notevoli capacità di comunicazione, storage e di computazione al fine di assicurare comunicazioni real-time o near-real-time; mentre per altri tipi di domini, in cui il requisito tempo non è così stringente, potrebbero essere sufficienti aggiornamenti orari, giornalieri o addirittura periodi più lunghi.
  In alcune circostanze come ad esempio il settore delle applicazioni mediche, robotica e settore industriale potrebbero essere necessarie tecniche e protocolli mirati ad assicurare il rispetto dei vincoli temporali.

- Tipo di associazione: è la tipologia di link presente tra l'asset fisico e il Digital Twin. Esso può essere unidirezionale o bidirezionale.
  Nelle prime versioni di Digital Twin, in cui la replica aveva l'unico scopo di rispecchiare lo stato attuale dell'asset fisico, era sufficiente una comunicazione unidirezionale dall'asset al Digital Twin. Questa tipologia assicurava corrette letture di sensori, degli eventi e quindi dello stato generale dell'asset e consentiva la sincronizzazione.
  Dopodiché sono nati Digital Twins con obiettivi diversi che presentavano una comunicazione unidirezionale dal Digital Twin all'asset. Questa consentiva di mandare comandi all'asset o azionare attuatori basandosi su informazioni provenienti da sorgenti esterne come ad esempio l'ambiente (si pensi al Digital Twin di un gazebo pergolato con chiusura automatizzata basata su dati raccolti dall'ambiente attraverso un sensore di pioggia). Oggigiorno invece, è ormai condiviso l'ovvio vantaggio di presentare una comunicazione bidirezionale tra il Digital Twin e l'asset fisico. Essa porta con sé numerose possibilità e servizi. Ad esempio, l'asset fisico fornirà tutti gli aggiornamenti di stato, gli eventi e i cambi di comportamento (al fine di effettuare la sincronizzazione), dopodiché il Digital Twin aggiornerà i dati associati al suo modello, li analizzerà (anche attraverso modelli di ML o AI) al fine di estrapolare insights o azioni correttive, preventive o migliorative e grazie al canale bidirezionale invierà comandi precisi e dettagliati all'asset fisico al fine di applicare le azioni decise.

Persistency

Come descritto dalla Figura 1.5 le applicazioni, al fine di ottenere i dati riguardanti gli asset, fanno riferimento (e richiesta) direttamente ai Digital Twins. Quindi, una proprietà fondamentale che devono presentare i Digital Twins è la *persistenza nel tempo*. Infatti, l'asset nel corso del tempo è caratterizzato dai suoi vincoli limitanti, mentre il Digital Twin deve sempre essere disponibile e accessibile rendendo di fatto la persistenza, la resilienza e

l'affidabilità concetti cardine dei Digital Twins e in particolare dell'eventuale piattaforma, framework o middleware che li gestisce.

Memorization

Un concetto forse non ancora adeguatamente sottolineato è l'importanza della *memorizzazione della storia* del Digital Twin e quindi del corrispondente asset fisico.

La storia (o digital thread [18]) è l'insieme dei dati raccolti e derivati dal Digital Twin. Ci consente di poter analizzare il comportamento, lo stato e gli eventi che hanno caratterizzato il passato e quindi poter usare questi dati, oltre che per capire il comportamento di un asset all'interno del suo spazio operazionale, per effettuare previsioni su possibili comportamenti all'interno dello stesso spazio o in altri ambienti (*what-if question*, *stress test* ...).

L'abilità di memorizzare e rappresentare i dati rilevanti (sottosezione Rappresentatività e contestualizzazione) e passati del Digital Twin è un concetto abilitante di cui è importante non privarsi al fine di poter adottare le recenti tecniche di analisi e prevenzione, molto importanti in contesti come la gestione delle città, l'healthcare o anche il manifatturiero.

Una questione di rilievo potrebbe essere la quantità di dati da prendere in considerazione rispetto al contesto e al dominio applicativo. La ricerca [15] suggerisce di sviluppare modelli in grado di rappresentare e memorizzare la maggior quantità di informazioni possibili in quanto potenzialmente utili per analisi future date dai progressi nelle tecniche e negli strumenti. Ovviamente, oltre alla quantità è di fondamentale importanza la *qualità* dei suddetti dati in quanto le applicazioni devono poter avere a disposizione dati affidabili e quindi di qualità.

Composability

Nel mondo reale possiamo considerare la maggior parte dei sistemi a due livelli di astrazione: attraverso una visione d'insieme in cui si vede il sistema come un singolo oggetto, oppure come il risultato di una combinazione di di-

versi oggetti.
Ad esempio, se prendessimo in considerazione un'auto la potremmo vedere come un unico oggetto, cioè un sistema che ha un suo stato, proprie caratteristiche e proprie azioni; oppure potremmo vederla come l'aggregazione delle singole componenti che interagiscono tra loro: l'impianto frenante, il motore, il cambio, le gomme ecc... In quest'ottica, il Digital Twin può essere visto come un *sistema di sistemi*. Infatti l'auto è un sistema che contiene altri sistemi, come appunto l'impianto frenante.

Perciò, con *Composability* si intende l'abilità di rappresentare un oggetto come la composizione di diversi oggetti con la possibilità di osservare e controllare il comportamento dell'oggetto composto allo stesso modo delle varie componenti.

I casi di *sistemi di sistemi* sono molto frequenti, si pensi agli edifici intelligenti, alle città, agli ospedali, alle fabbriche ...

Sarà compito dello sviluppatore e del progettista capire, a seconda del contesto e del dominio applicativo, il livello di complessità e composizione a cui arrivare al fine di rappresentare l'asset all'interno dello spazio virtuale.
Inizialmente, si potrebbe rappresentare il sistema nel suo complesso ed inserito all'interno dell'ambiente virtuale; dopodiché, con l'avanzare delle informazioni a disposizione si potrebbe pensare di provvedere un modello più completo che rappresenti anche le varie componenti e le relazioni presenti tra di esse, fino a descrivere completamente il sistema o una sua sottoparte come l'insieme di varie parti, ognuna delle quali osservabili ed azionabili.

Quindi, i framework e le piattaforme a supporto dei Digital Twins devono assicurare integrazione e componibilità affinché sia possibile modellare correttamente i sistemi complessi.

Accountability/Manageability

Dalle proprietà già evidenziate, si evince la necessità di gestire in modo preciso ed accurato i Digital Twins. L'*Accountability* e la *Manageability* si

riferiscono proprio alla capacità di controllare, governare i Digital Twins e mantenere il link tra lo spazio virtuale e quello reale.

Ad esempio l'Entanglement è una proprietà che porta con sé notevoli difficoltà di gestione. Si potrebbe verificare la situazione in cui l'asset fisico abbia un problema di comunicazione oppure si danneggi. In questo caso, la sua controparte digitale (il Digital Twin) non può assolutamente andare offline, ma anzi, deve entrare in uno stato di recovery in cui continua a soddisfare le richieste delle applicazioni fornendo i dati storici oppure l'ultimo stato registrato. Invece, in caso di problema della piattaforma a supporto dei Digital Twins per cui si perde il link tra l'asset fisico e la sua replica è necessario individuare un insieme di procedure con cui ripristinare e riconfigurare velocemente la comunicazione con la minima perdita di informazione.
Affinché il middleware venga considerato affidabile sono necessarie tecniche all'avanguardia di gestione e orchestrazione.

Inoltre, i Digital Twins possono essere utilizzati contemporaneamente da più applicazioni, perciò è molto importante che vengano applicati tutti i meccanismi adeguati di virtualizzazione e scalabilità lato piattaforma.

Augmentation

L'IoT ha già presentato una via per l'aumento delle potenzialità e delle funzionalità di un asset fisico rappresentando via software i suoi dati. Infatti, i dati raccolti tramite i sistemi IoT possono essere integrati per fornire insights, spediti ad un qualche modello di ML per fare previsioni o più semplicemente richiesti da un'entità esterna.

Seguendo lo stesso filo conduttore, modellando un aspetto della realtà attraverso un modello software, si aprono un insieme di strade infinite nell'elaborazione e nella fornitura di servizi e funzionalità aggiuntive attraverso l'uso del solo software. Infatti, gli asset fisici vengono creati con un insieme ben definito di funzionalità che li contraddistinguono durante tutto il ciclo di vita. Invece, grazie ad una rappresentazione virtuale dell'asset fisico è possibile inserire un

layer di astrazione sopra al mondo reale che può essere aumentato a piacere in quanto basato unicamente sul software.

I Digital Twins possono contare sulla dematerializzazione del software in modo da poter modificare, aggiornare e migliorare le proprie funzionalità nel tempo [15]. Il Digital Twin di un asset potrebbe avere delle funzioni aggiuntive che la controparte fisica non presenta in nessuna versione. Può fare uso dell'intero stato dell'asset oppure di dati provenienti dalla collaborazione tra più Digital Twins al fine di fornire un servizio aggiuntivo allo spazio virtuale (il quale si collega a quello reale). Anche l'oggetto più statico che esista, come ad esempio una statua, attraverso il suo Digital Twin potrebbe fornire qualche servizio e quindi cooperare all'interno di un ambiente.

Ovviamente, al fine di mettere a disposizione le features aumentate è necessario inserire un layer di comunicazione che solitamente è rappresentato da un insieme di APIs con il quale gli asset fisici, attraverso le proprie controparti virtuali, possono essere resi interoperabili all'interno di un ambiente complesso e collaborare per raggiungere obiettivi comuni. Infatti, anche se due asset fisici utilizzassero un protocollo diverso all'interno del mondo reale, sfruttando il software sarebbe possibile portarli a "parlare la stessa lingua".

Ownership

Quando si parla di dati presenti sul cloud o comunque all'interno di un sistema IT, occorre trattare l'argomento *Ownership*, cioè *proprietà dei dati*. I dati all'interno di un sistema di Digital Twin rappresentano la maggiore e principale forma di informazione: senza dati non possono esistere i Digital Twins. Quindi, in conformità di tutte le leggi e i regolamenti attualmente in vigore (primo esempio su tutti il GDPR[6]) è necessario stabilire il proprietario dei dati memorizzati rispetto ad un Digital Twin. Inoltre, è allo stesso modo importante la proprietà dei Digital Twins stessi.

[6] `https://ec.europa.eu/info/law/law-topic/data-protection/data-protection-eu_it`

Per capire questa seconda tipologia di proprietà basta prendere come esempio un quadro. Un quadro viene dipinto da un pittore il quale nella maggior parte dei casi è anche il proprietario dei diritti patrimoniali (quindi il proprietario che è in grado di sfruttarlo economicamente). Di quel quadro è possibile creare un Digital Twin che può essere dato in licenza ad un museo oppure copiato e modificato andando a creare un opera considerata nuova, creativa e originale, perciò tutelabile dalla legge sul diritto d'autore. Questo fa si che sia necessario avere la possibilità di stabilire il proprietario di un singolo Digital Twin in ogni istante temporale, il quale appunto può essere diverso dal proprietario della controparte fisica originale. Un ulteriore esempio dell'importanza del tracciamento delle proprietà è il caso di un auto usata. Essa nel tempo può avere diversi proprietari e quindi al fine di rappresentare completamente lo stato dell'auto è necessario andare ad effettuare il tracking di quest'ultimi. In questo modo sarà possibile fornire anche informazioni aggiuntive come ad esempio lo stato generale dell'auto nel momento del passaggio di proprietà, la distanza percorsa ecc..

La gestione dell'*Ownership* è sicuramente un aspetto molto complesso da gestire in modo affidabile e sicuro, però è un ulteriore aspetto da tenere in considerazione nella creazione di un'architettura a supporto dei Digital Twins.

Servitization

Legata in particolare all'Augmentation, la *Servitization* riguarda la possibilità di offrire sul mercato un prodotto arricchito di funzionalità, servizi, processi e accesso ai dati attraverso software, strumenti e interfacce. Questo concetto introduce il mezzo con cui creare nuovi servizi che possono operare sull'asset fisico direttamente dal Digital Twin.

Tutto ciò è abilitato dalle proprietà descritte precedentemente. Infatti, tramite la rappresentatività, contestualizzazione, riflessione e memorizzazione si è in grado di memorizzare le informazioni di stato del Digital Twin. Dopodiché grazie all'Entanglement è possibile mantenere un canale bidirezionale continuo che può essere sfruttato attraverso l'Augmentation al fine di fornire

l'interfaccia per richiamare i vari servizi.

In questo modo è possibile vendere una lampadina con associato il proprio Digital Twin in modo da poterne controllare lo stato e poterla accendere o spegnere direttamente dalla sua replica virtuale o, in caso di Cognitive Digital Twin, può essere comandata automaticamente dalla propria replica virtuale raccogliendo e analizzando informazioni provenienti da altri strumenti o direttamente dall'ambiente stesso al fine di intraprendere le azioni migliori rispetto al contesto.

Quindi si può ben capire come lato business i Digital Twins rappresentino un trampolino di lancio. Le aziende possono mantenere un link diretto e continuativo con il prodotto e quindi con il cliente, con la possibilità di fornire manutenzione preventiva e con la creazione di servizi a pagamento con diversi piani associati a seconda del prezzo. La *Servitization* sta ponendo le basi per un cambiamento nella vendita di prodotti, i quali ora necessitano sempre di più servizi associati e accessibili da remoto, spingendo l'economia basata sulla proprietà di prodotti, ad una economia basata sul "pay per usage" [15].

Predictability

I sistemi di Digital Twins processano e producono enormi quantità di dati riguardanti principalmente proprietà ed eventi provenienti dalla controparte fisica. Questi dati possono essere sfruttati per individuare pattern, anomalie e più generalmente insights oppure per capire il funzionamento dell'asset associato (assieme al modello di comportamento) all'interno di in un ambiente simulato.

La *Predictability* è la proprietà dei Digital Twins di poter essere inseriti all'interno di svariati ambienti al fine di eseguire simulazioni sul comportamento e sulle interazioni con gli altri oggetti in situazioni particolari o di interesse. Inoltre, come detto, grazie alla notevole quantità di dati a disposizione, è possibile utilizzare le recenti tecniche offerte dall'AI per effettuare previsioni dello stato o del comportamento futuro.

Questi vantaggi possono essere utilizzati in tutti i contesti, ad esempio: gestione del traffico, previsione di malessere di un paziente o previsione dell'efficacia di una cura, simulazione e previsione del momento migliore per la somministrazione di un medicinale basandosi sullo stato corrente e futuro (predetto) di un paziente ecc...

Riepilogo

Alcune delle proprietà elencate erano già presenti nella versione di base elaborata da Grieves nel 2003 [9][10] quando il compito principale del Digital Twin era il mirroring dello stato e del comportamento di un asset fisico. Queste sono: Rappresentatività, Contestualizzazione, Riflessione, Virtualizzazione e Entanglement.

Tutte le restanti proprietà sono state introdotte grazie al survey di Minerva, Lee e Crespi e rappresentano la base di partenza della visione offerta da questa tesi nel Capitolo 4.

La Figura 1.6 offre uno schema riassuntivo e diretto delle principali proprietà dei Digital Twins elaborate da Minerva, Lee e Crespi.

1.4 Verso la standardizzazione

Il concetto di Digital Twin trova spazio non solo nel mondo manifatturiero, ma in qualsiasi settore in cui è applicabile un clone software aumentato di un asset fisico. Dal punto di vista concettuale, la ricerca e le industrie offrono notevoli punti di vista e tanto materiale da analizzare e studiare. Però, ancora oggi, non vi è una visione condivisa a cui si possa fare riferimento al fine di implementare una soluzione omogenea ed interoperabile.

Nel maggio del 2020, Microsoft e altri co-fondatori hanno creato un primo consorzio al fine di stabilire degli standard in materia di Digital Twin: il *Digital Twin Consortium*, che verrà descritto nella sottosezione 1.4.1.

Inoltre, recentemente il concetto di Digital Twin è d'interesse anche per alcune nazioni, come il Regno Unito in cui si stanno creando delle vere e proprie

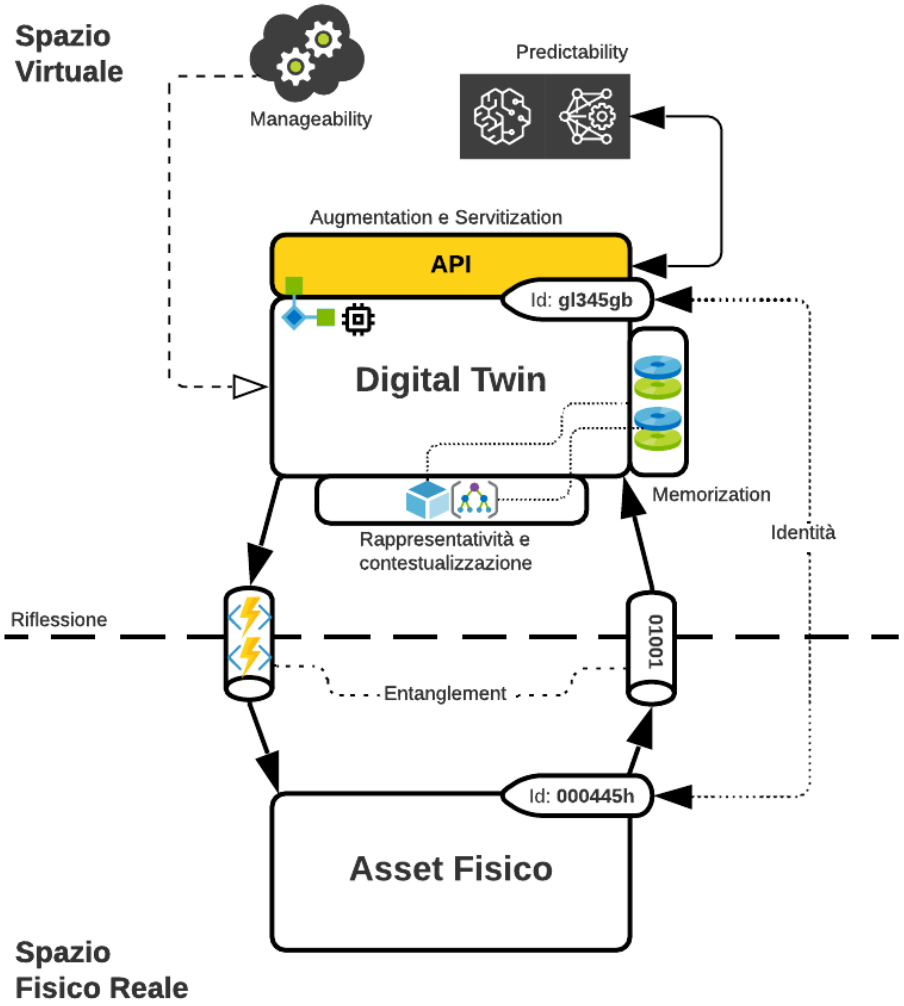

Figura 1.6: Rappresentazione delle principali proprietà dei Digital Twins

iniziative per trasformare aspetti di interesse della nazione attraverso l'impiego dei Digital Twins. In particolare, nella sottosezione 1.4.2, verrà descritto il *National Digital Twin programme*, l'iniziativa del Regno Unito per la creazione di un ecosistema di Digital Twin interconnessi per lo scambio di dati.

1.4.1 Digital Twin Consortium

Il *Digital Twin Consortium (DTC)* fu annunciato il 18 Maggio 2020 e cofondato da Microsoft, Dell, Ansys, Autodesk, GE Software, Northrop Grumman, and Lendlease assieme al consorzio per gli standard del settore informatico OMG[7]. Il DTC oggi conta più di 170 membri che spaziano tra aziende, agenzie governative ed accademie che assieme lavorano alla consistenza nel vocabolario, nell'architettura, nella sicurezza e nella compatibilità e interoperabilità dei Digital Twins. L'obiettivo finale del DTC è la creazione di standard su scala mondiale che permettano l'utilizzo e la condivisione cross-organizzazione delle informazioni prodotte dai Digital Twin per aiutare a far progredire l'uso di questa tecnologia in molti settori, dall'aerospaziale alle risorse naturali.

Per questo motivo, il Digital Twin Consortium è attualmente (o si considera) "l'Autorità" nei Digital Twin in quanto è l'unica autorità che attualmente lavora per accelerare lo sviluppo dei Digital Twin e per stabilire degli standard nelle tecnologie, nelle definizioni oltre che pubblicare riferimenti per lo sviluppo di framework e software open-source[8] al fine di massimizzare i benefici e l'interoperabilità.

Uno dei principali problemi dell'applicazione dei Digital Twins all'interno delle industrie è la presenza di silos verticali in cui il concetto di Digital Twin non trova una definizione comune. Per questo motivo, il DTC ha elaborato un glossario contenente tutti i concetti principali che compongono il mondo dei Digital Twins oltre che a fornire un framework di riferimento ad alto livello (analizzato nel Capitolo 2).

Il glossario[9] è un lavoro ancora in corso e si pone l'obiettivo di stabilire degli standard nella definizione dei principali concetti del mondo dei Digital Twins. Non si concentra minimamente sui vari domini industriali, si limita solamente ai concetti generali.

La prima e più importante definizione che troviamo è quella di Digital Twin:

[7] Object Management Group `https://www.omg.org/`

[8] `https://www.digitaltwinconsortium.org/initiatives/open-source.htm`

[9] `https://www.digitaltwinconsortium.org/glossary/index.htm`

> *A digital twin is a virtual representation of real-world entities and processes, synchronized at a specified frequency and fidelity.*

Inoltre aggiunge:

> *Digital Twins use real-time and historical data to represent the past and present and simulate predicted futures.*

Si può notare come venga introdotto il concetto di *fidelity* e di *frequenza di sincronizzazione*. Queste sono caratteristiche di fondamentale importanza se si pensa ad una reale implementazione; perciò, verranno considerate successivamente.

Inoltre, vediamo come anche i processi vengano inclusi all'interno della definizione. Questo consente di poter rappresentare anche operazioni di particolare interesse all'interno del contesto (ad esempio un intervento chirurgico) sfruttando tutti i vantaggi dell'approccio a Digital Twins.

1.4.2 National Digital Twin programme

Il Centre for Digital Built Britain (CDBB) in partnership con l'Università di Cambridge e il dipartimento di business, energia e strategia industriale gestisce il National Digital Twin programme[10] (NDTp) con l'obiettivo di creare un sistema di Digital Twin connessi a livello nazionale.

Il National Digital Twin programme fu lanciato nel Luglio 2018 da HM Treasury[11] con lo scopo principale di portare tutti i vantaggi dell'approccio a Digital Twin all'interno dello stato:

- Benefici sulla società: i Digital Twins consentono di sviluppare nuovi servizi e migliorare quelli esistenti. Ciò porta vantaggio a qualsiasi livello della società, partendo dai trasporti fino agli ospedali.

[10] `https://www.cdbb.cam.ac.uk/what-we-do/national-digital-twin-programme`

[11] *Her Majesty's Treasury* è un Dipartimento governativo del Regno Unito responsabile per lo sviluppo e l'esecuzione delle politiche di finanza pubblica e per la politica economica del governo britannico. Da Wikipedia `https://it.wikipedia.org/wiki/HM_Treasury`

- Vantaggi economici: grazie alla piattaforma di condivisione dati, i Digital Twins possono condividere insights utili ad aumentare la produttività generale delle industrie e delle infrastrutture inglesi. Inoltre, grazie alle nuove tecnologie di sicurezza informatica tutto ciò viene fornito assicurando una notevole sicurezza.

- Nuovi business: come visto prima, i Digital Twins sono una chiave abilitante per nuovi business e nuovi servizi. Inoltre, grazie alla Servitization, vanno anche ad aumentare le entrate per i servizi già esistenti, grazie alla possibilità di fornire nuovi piani di utilizzo che dipendono interamente dal software.

- Vantaggi per l'ambiente: grazie alla condivisione delle informazioni è possibile gestire in miglior modo l'inquinamento, i rifiuti e lo spreco di qualsiasi materia prima o alimentare. È possibile monitorare il consumo energetico e fornire piani che vadano ad ottimizzare l'uso delle risorse. Tutto ciò è di fondamentale importanza per combattere il cambiamento climatico.

Il CDBB con il National Digital Twin non intende creare un unico grande Digital Twin in grado di modellare ogni aspetto della nazione, sarebbe impensabile. Invece, vuole creare un ecosistema di Digital Twins distribuiti in federazioni ed interconnessi da un layer sicuro di condivisione dati.

I Digital Twins possono trovarsi a diversi livelli di scala o grado: locale, regionale o nazionale. Quindi, il termine *National* Digital Twin, visto che non si può riferire al fatto di avere un unico Twin a livello nazionale, descrive il risultato dell'applicazione di un *approccio a livello nazionale* per la gestione delle informazioni all'interno dell'*ecosistema di Digital Twin*.
I Digital Twins potranno far parte di domini e organizzazioni completamente diversi. Questo sarà uno dei punti di forza del sistema in quanto potrà lavorare con piattaforme e middleware per Digital Twin differenti, assicurando la condivisione sicura di dati. Per questo motivo, non ci sarà una data di "fine progetto" in quanto l'NDTp è un progetto vivente in simbiosi con l'ambiente.

Ovviamente, non sarà necessario prevedere connessioni uno a uno tra tutti i Digital Twins, ma saranno connessi solo quelli per cui la condivisione di informazioni porterà un valore aggiunto. Infatti, lo slogan principale che muove tutto il progetto è *Data for the public good*.

Da uno studio di Novembre 2017 [7] e da statistiche di Agosto 2018 [16] si evince che una migliore gestione della condivisione dei dati può portare ad un beneficio annuo di 7 miliardi di sterline in tutto il settore delle infrastrutture inglese, l'equivalente del 25% delle spese totali. Il valore dei dati cresce esponenzialmente appena questi vengono aggregati e condivisi tra le organizzazioni.
Una gestione più efficiente e soprattutto più efficace delle informazioni abilita a migliori decisioni in tutti i settori dell'economia e della società.
Basti pensare ai settori dell'energia e dei trasporti i quali, individualmente, sviluppano le proprie soluzioni basate su Digital Twin.

Integrare dati inter e intra settore consente di scovare nuove informazioni, nuovi insights e permette di elaborare modelli in un modo che, con le tecnologie basate sulla singola organizzazione, non sembrava neanche possibile.

Al fine di consentire questa gestione delle informazioni è necessario sviluppare un framework capace di definire le regole e controllare la condivisione dei dati. Questo framework si chiama *Information Management Framework (IMF)* e verrà descritto in seguito.

Prima di tutto però, occorre definire una serie di principi sulla base dei quali costruire l'IMF, l'NDT e permettere la condivisione di dati. Questi principi sono stati sviluppati dal Digital Framework Task Group e dal CDBB e vengono chiamati *Gemini Principles*.

Gemini Principles

Per rendere il National Digital Twin funzionale, affidabile e utile sono necessari dei forti principi fondanti che guidino lo sviluppo e creino un punto di allineamento nella condivisione dei dati. Questi principi vengono chiamati *Gemini Principles* [11].

I Gemini Principles sono nove e si limitano ad offrire una descrizione dell'intento rimanendo completamente agnostici sulla particolare soluzione adottata. Descrivendo solo il concetto e tralasciando l'aspetto implementativo consentono una grande elasticità e flessibilità nell'innovazione e nell'aggiornamento delle tecnologie utilizzate. Essi sono organizzati in tre titoli principali: Purpose, Trust e Function.

Purpose rappresenta la necessità di ogni parte del National Digital Twin di avere un chiaro scopo oltre a quello generale di migliorare i risultati per "whole-life pound". Racchiude sotto di sé tre principi:

- Public good: l'IMF e l'NDT sono risorse a livello nazionale, perciò devono essere di pubblica utilità.

- Value creation: le informazioni condivise all'interno dell'ecosistema di Digital Twins devono abilitare la creazione di un valore aggiunto come il miglioramento di performance o la gestione di rischi. Questo consentirà di avere un ritorno economico e un miglioramento della produttività a livello nazionale.

- Insight: i Digital Twins, all'interno dell'NDT, collaborando devono dare la possibilità di estrapolare nuovi insights i quali, assieme ai dati, contribuiscono ad ottenere un miglioramento nelle decisioni di pubblico interesse.

Trust rappresenta la necessità di ogni parte del National Digital Twin di essere affidabile e sicura in modo da consentire a tutti i proprietari di asset o di Digital Twins di condividere informazioni in modo protetto. Racchiude sotto di sé tre principi:

- Security: la sicurezza informatica e la privacy giocano un ruolo centrale anche per l'NDT. È necessario assicurare la protezione e la privacy dei dati personali, dei dati sensibili delle infrastrutture a livello nazionale oltre che proteggere il sistema per quanto riguarda le proprietà intellettuali e i rischi di consistenza dei dati aggregati.

- Openness: l'NDT deve utilizzare quante più sorgenti open possibili: open data, open culture, open standard e tecniche open source, rispettando sempre il principio di Security. L'impiego di elementi open aumenta la fiducia nel sistema, diminuisce i costi e crea più valore in quanto aumenta l'interoperabilità.

- Quality: i dati condivisi all'interno dell'NDT devono essere di una qualità adeguata all'obiettivo per il quale sono stati raccolti e condivisi. La qualità non si sofferma solamente sulla pulizia dei dati, ma anche sulla sicurezza e sulla longevità del dato rispetto al contesto di utilizzo. Ad esempio, in campo medico nella maggior parte dei contesti i dati devono essere condivisi in real-time, altrimenti diventano insignificanti e quindi non di qualità per il contesto designato.
  Sono necessari degli standard e dei modelli in grado di descrivere i requisiti di ogni contesto in modo da poter stabilire all'interno dell'NDT la qualità e l'accuratezza dei dati richiesti. Il framework stesso perciò deve supportare la memorizzazione di attributi riguardanti la qualità.

Function rappresenta la necessità del sistema di essere sempre funzionante e disponibile per far fronte a qualsiasi richiesta dell'utente finale. Racchiude sotto di sé tre principi:

- Federation: come detto precedentemente, l'NDT è un ecosistema di Digital Twins organizzati in federazioni. Le federazioni devono essere in grado di comunicare attraverso un layer di condivisione dati sicuro e standardizzato. L'interoperabilità è di fondamentale importanza in questo contesto in quanto i Digital Twins provengono da organizzazioni sparse per tutta la nazione e quindi l'NDT deve essere in grado di lavorare con diversi Twin Builder, diversi provider e diversi tipi di dati provenienti da domini diversi.

- Curation: qui rientra il principio di Ownership descritto nella sezione precedente. Ogni tipologia e set di dati deve indicare il proprietario, il

quale deve assicurare il livello di qualità indicato. Inoltre, sono necessari meccanismi per controllare che gli standard vengano rispettati e per regolare il flusso di dati.

- Evolution: l'NDT deve astrarre dai dettagli implementativi e soprattutto dalle particolari tecnologie adottate nelle soluzioni, nelle piattaforme e in tutto il software in generale. Questo consente al framework di adattarsi ai cambiamenti tecnologici non appena essi si presentano, rimanendo in linea con l'evoluzione IT. In generale, deve poter essere espanso e modificato dinamicamente nel tempo, mantenendo la propria operatività.

Information Management Framework

Ad oggi pochissimi Digital Twins sono connessi o condividono informazioni cross-organizzazione. Questo perché vi è un gap di interoperabilità tra le diverse soluzioni.

L'obiettivo dell'*Information Management Framework* (IMF) è di stabilire un linguaggio comune attraverso il quale Digital Twin appartenenti ad organizzazioni e ambienti diversi possano comunicare in modo sicuro ed essere uno strumento di supporto efficiente per il processo di decision making e per i servizi che caratterizzano la società.

L'IMF consente l'interoperabilità dei dati in modo sicuro e resiliente rispettando la privacy e i Gemini Principles. L'IMF è quindi il cuore del National Digital Twin (NDT).

Perciò, quello che si vuole creare attraverso l'NDT e l'IMF non è un Twin Builder, cioè una piattaforma capace di creare Digital Twins ed assicurare tutte le proprietà e le caratteristiche come descritto nella Sottosezione 1.3.1, ma un framework capace di mettere in comunicazione Twin Builder esistenti, diversi e appartenenti ad organizzazioni e domini diversificati (Figura 1.7).

Questo concetto, molto più di alto livello ed espansivo rispetto a ciò che ho descritto fin'ora, abilita i Digital Twins a pattern molto più complessi di utilizzo rispetto alle semplici query sullo stato. Infatti, un esempio riportato in [12] è quello del seguente problema: *"Quali grattacieli nel Regno Unito*

hanno i tipi di rivestimento identificati nell'inchiesta Hackitt come ad alto rischio di incendio? Dopodiché classificali in base al rischio." Per rispondere ad una domanda del genere è necessario uno sforzo a livello di esplorazione ed integrazione di informazioni molto importante. È necessario infatti: determinare i grattacieli con quel determinato tipo di rivestimento tra tutti i grattacieli memorizzati attraverso i modelli strutturali e architetturali, ottenere i dati esistenti del rischio incendio, ottenere i dati dell'occupazione degli edifici rispetto al tempo, combinare questi dati con le procedure di evacuazione e con i modelli di predizione sulla prontezza dei veicoli di emergenza ed infine comporre tutti questi dati e modelli attraverso un modello matematico al fine di ottenere una classifica degli edifici sulla base del livello di rischio incendio. Solamente se le informazioni sono accessibile in modo semplice, consistente e resiliente è possibile risolvere una query del genere in modo agevole, e l'IMF punta a ciò.

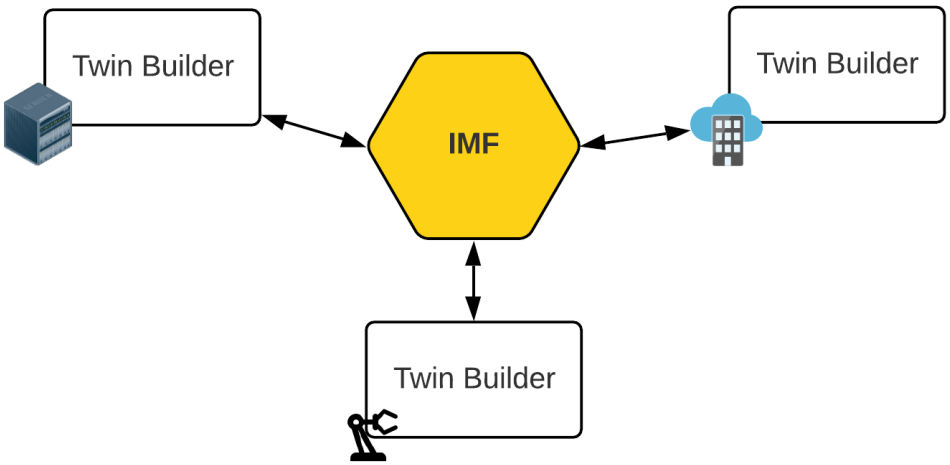

Figura 1.7: Visione astratta dell'Information Management Framework

Nel Capitolo 2 descriverò i concetti principali che sono dietro all'architettura dell'Information Management Framework e del National Digital Twin in generale.

1.4.3 Riepilogo

Il Digital Twin Consortium assieme al National Digital Twin programme rappresentano due delle principali iniziative che hanno l'obiettivo di portare forme di standardizzazione all'interno del mondo dei Digital Twins.

È necessario ancora tanto lavoro da parte delle aziende e sopratutto da parte della ricerca per cercare di dare una forma unica e condivisa del concetto.

Un passo importante ed urgente è lo sviluppo di un framework per la creazione di Digital Twins che vada ad implementare tutte le caratteristiche e i concetti descritti in questo capitolo.

I prossimi due capitoli verteranno la propria attenzione alle soluzioni attualmente sul mercato e anche ad alcune visioni promosse dal National Digital Twin programme e dal Digital Twin Consortium con i rispettivi IMF e Digital Twin System.

Capitolo 2

Proposte architetturali

Dopo aver descritto le caratteristiche e i diversi punti di vista che rappresentano il concetto odierno di Digital Twin, al fine di sviluppare una propria visione e una propria idea di middleware, occorre avere una panoramica sulle principali proposte architetturali che la ricerca e le organizzazioni hanno sviluppato durante questi anni.

Come si vedrà, le soluzioni proposte confermano la mancanza di standardizzazione e di condivisione di una definizione unica di Digital Twin. Infatti, ogni azienda, accademia o agenzia governativa considera solamente quelle caratteristiche che, ovviamente, rientrano all'interno della propria visione.

Le soluzioni descritte in questo capitolo sono, ad oggi, solo a livello concettuale: il "Digital Twin System" ideato dal Digital Twin Consortium, l'Information Management Framework ideato nel corso dell'iniziativa "National Digital Twin programme" o le proposte provenienti dalla ricerca [6] [17], come tante altre, non hanno ancora completato il proprio ciclo di sviluppo e in alcuni casi non è neanche loro intenzione completarlo al fine di rimanere una linea guida architetturale per poter sviluppare nuove piattaforme (come nel caso probabilmente del Digital Twin System).

2.1 DTC Digital Twin System

Il Digital Twin System è l'architettura proposta all'interno del glossario dal Digital Twin Consortium (DTC)[1].

Il DTC vede il Digital Twin System come un sistema di sistemi volto all'implementazione di un Digital Twin. Esso è composto da diverse componenti come illustrato nella Figura 2.1:

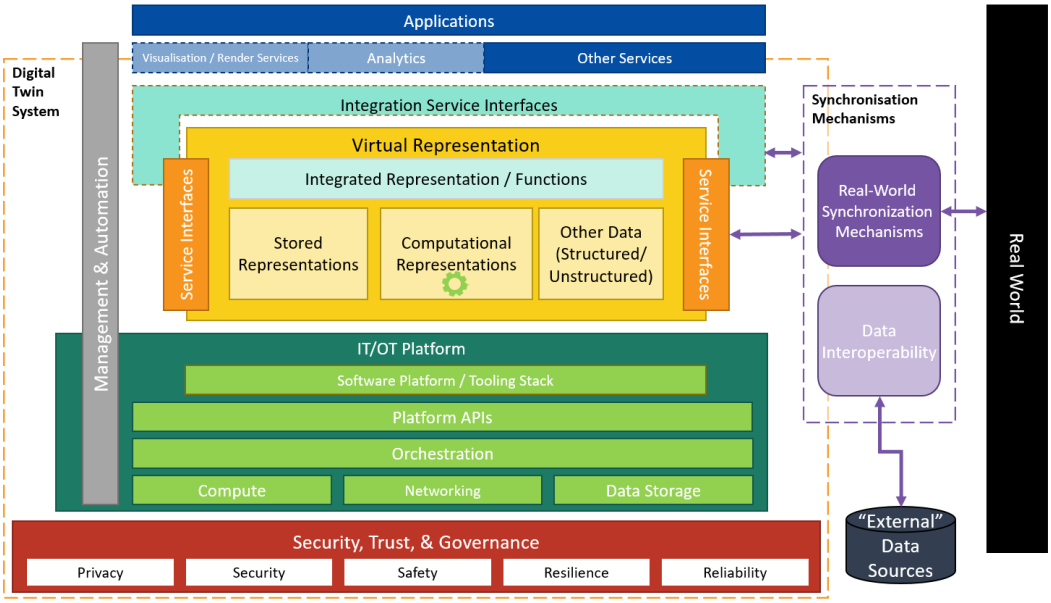

Figura 2.1: Digital Twin System

Innanzitutto, si può notare come vi sia una netta separazione tra lo spazio virtuale e il mondo reale. Al fine di mantenere aggiornata la controparte digitale sono necessari dei meccanismi che il DTC chiama "Synchronization Mechanisms" i quali prevedono anche un layer di interoperabilità per la comunicazioni con più sorgenti di dati. Inoltre, da sottolineare il fatto che il canale di comunicazione è bidirezionale in modo tale da poter inviare comandi al mondo fisico direttamente dalla replica.

[1] `https://www.digitaltwinconsortium.org/glossary/index.htm#digital-twin-system`

Il Digital Twin System è formato da diversi componenti i quali, si suppone, siano individualmente scalabili in modo da adattarsi al contesto di utilizzo.
Partendo dal basso si nota un layer che si occupa della gestione della sicurezza, della privacy, della resilienza e quindi in generale della disponibilità del sistema. Essa è di fondamentale importanza per assicurare la proprietà di Persistency.
A questo si appoggia il reparto di IT/OT che si occupa di tutta l'infrastruttura di Information Technology e Operational Technology su cui il Digital Twin System è implementato. L'infrastruttura è composta da sottosistemi come ad esempio: APIs, sistema per la gestione della replica (orchestrazione) e tutte le risorse computazionali, di rete e di storage. Questi sottosistemi possono trovarsi ovunque: in cloud, embedded o addirittura essere distribuiti.
Il cuore del Digital Twin è contenuto all'interno del componente chiamato "Virtual Representation". Esso si occupa della memorizzazione dei dati, degli aspetti computazionali (simulazione, non behavior) e di esporre tutto ciò all'esterno per l'utilizzo tra sistemi e tra organizzazioni. I dati principali vengono memorizzati all'interno del componente "Stored Representation". Esso contiene tutti i dati strutturati che rappresentano il modello permettendo l'esecuzione di query, inserimenti ed aggiornamenti.
Inoltre, in questa visione offerta dal DTC anche la parte di simulazione e analisi predittiva è interna al Digital Twin (in particolare all'interno del componente "Computational Representations") e la sua esecuzione può essere richiesta dalle stesse interfacce utilizzate per richiedere i dati.
Abbiamo due tipi principali di interfacce: Service Interface e Integration Service Interfaces. La prima consente di dialogare con altri sistemi o servizi; la seconda invece per fornire l'accesso al Digital Twin tramite l'Integrated Representation/Functions la quale offre un ulteriore vista sulla replica.

Il Digital Twin System è quindi una visione architetturale che si pone ad un livello molto astratto, concettuale, senza scendere in dettagli implementativi e non trova infatti ancora un'implementazione ufficiale. Ciò che è possibile ad oggi è creare un'architettura che sia conforme con quella proposta internamente al Consorzio.

2.2 National Digital Twin

Il *National Digital Twin programme*, creato dal Centre for Digital Built Britain in partnership con l'Università di Cambridge, rappresenta un ottimo caso di studio per l'analisi delle piattaforme esistenti a supporto dei Digital Twins. Come descritto precedentemente, il NDT non si pone come obiettivo la creazione di un Twin Builder a livello nazionale, bensì la creazione di un ecosistema di Digital Twins appartenenti ad organizzazioni diverse raggruppati in federazioni. Per fare ciò si pone ad un livello superiore rispetto alle varie piattaforme e ai framework oggi in commercio, cercando di creare un layer tramite il quale fornire interoperabilità tra tutte le soluzioni. Al fine di raggiungere questo obiettivo è necessario un framework che, ad oggi, è pubblicamente fornito solo in forma concettuale e quindi in attesa di sviluppi concreti. Questo framework prende il nome di *Information Management Framework (IMF)*.

Nella capitolo precedente, ho dato una visione più teorica di cosa rappresenta l'IMF all'interno del NDT, qui invece mi concentrerò più sul punto di vista architetturale e fornirò una visione di alto livello sullo stato attuale dello sviluppo del progetto.

Innanzitutto, alla base dell'IMF vi sono i Gemini Principles i quali rappresentano le linee guida per la creazione di un ecosistema di Digital Twins sicuro ed innovativo.

Il CDBB in [12] e in [13] ha stabilito che l'IMF è formato da tre parti principali: *Foundation Data Model (FDM)*, *Reference Data Libray (RDL)* e *Integration Architecture (IA)*. L'IA rappresenta l'effettiva piattaforma che abilita lo scambio dei dati, i quali, per poter essere condivisi, devono utilizzare un linguaggio comune rappresentato dalla combinazione del FDM e del RDL.

Grazie agli standard per l'interoperabilità e per lo scambio di informazione, offerti da FDM, RDL e IA, NDT mira alla creazione di un *web of Digital Twins*, simile a ciò che già accade con l'Internet of Things.

Il termine "web of digital twin" rappresenta pienamente il concetto di National Digital Twin: un ecosistema distribuito di Digital Twins, individuabili attraverso i protocolli definiti, riuniti a seconda del bisogno per poter essere

interrogati ed azionati. Per arrivare a ciò è necessario creare una quantità di standard importante che spazi dalla definizione di modelli interoperabili e univoci all'interno dell'ambiente fino alla contestualizzazione nello spazio, tempo e nella società (con aspetti anche legali) dei dati prodotti dai vari asset e resi disponibili all'interno dell'ecosistema. Inoltre, il framework dovrà fornire protocolli per:

- Fornire il servizio di streaming dei dati in real-time o near-real-time.
- Consentire l'osservazione di un Digital Twin tramite pattern pubblish/-subscribe
- Gestire aspetti di sicurezza
- Gestire il controllo degli accessi: necessario per poter definire quali attori e con quali ruoli possono accedere alle informazioni prodotte dai Digital Twins o richiamare operazioni su di essi.

A tal fine, come anticipato, l'IMF viene suddiviso in tre parti principali: Foundation Data Model, Reference Data Library ed Integration Architecture.

Il Foundation Data Model rappresenta la *top-level ontology* [2] e fornisce una visione consistente delle caratteristiche fondamentali dei Digital Twins come abitanti dell'ecosistema. Essendo una top-level ontology è completamente indipendente dal dominio applicativo in quanto si sofferma su questioni e caratteristiche valide in tutti i contesti. Il suo obiettivo è permettere l'interoperabilità semantica di base tra tutte le organizzazioni e i Twin Builder.
FDM si occupa di questioni cruciali all'interno del framework che ritrovano spesso una corrispondenza con i Gemini Principles. I dati devono essere disposti all'interno dello spazio-tempo e devono essere associati ai requisiti di qualità al fine di poter comprendere la validità di essi all'interno del contesto d'analisi. Lo spazio-tempo e la qualità sono concetti presenti in qualsiasi dominio applicativo, perciò trovano perfettamente spazio nella definizione di

[2] chiamata anche "Upper Ontology": `https://en.wikipedia.org/wiki/Upper_ontology`

FDM. Quindi, Il Foundation Data Model include una definizione e una struttura della descrizione dello spazio e del tempo oltre che un insieme di attributi che vanno a rappresentare la qualità (e i suoi requisiti). Questo è in linea con i Gemini Principles, in particolare con il principio Quality, interno a Trust, che evidenzia proprio la necessità di associare ai dati i requisiti di qualità, lo spazio e il tempo. Altri concetti presenti in qualsiasi dominio e rientranti all'interno dei Gemini Principles sono: risoluzione dei problemi, versioning dei modelli ed ownership; essi trovano tutti spazio all'interno del Foundation Data Model. Per quanto riguarda i problemi che possono sorgere, esso prevede strategie per la risoluzione di conflitti tra più sorgenti di dati e per i conflitti che possono sorgere nella classificazione di un Digital Twin. Mentre, per quanto riguarda il versioning e l'ownership non sono ancora state proposte delle soluzioni in merito, però sono state prese in considerazione all'interno della modellazione.

La Reference Data Library rappresenta l'insieme delle ontologie specifiche di dominio. In particolare, viene definito come il "vocabolario delle classi e delle proprietà" che utilizzeremo per descrivere i Digital Twins in modo standardizzato all'interno del proprio contesto o dominio. Grazie a queste ontologie sarà possibile ottenere dati riguardanti uno specifico contesto da qualsiasi organizzazione collegata al National Digital Twin in quanto essi avranno la stessa struttura indipendentemente da dove verranno presi. Questo consente di ottenere una grande consistenza intra e inter settoriale permettendo lo scambio di dati in qualsiasi modalità. Inoltre, attraverso RDL vengono stabiliti anche i dati obbligatori e quelli aggiuntivi per un determinato concetto, assicurando così una quantità minima di informazione sempre presente qualsiasi sia la sorgente dei dati, spingendo l'interoperabilità a livelli sempre maggiori.

L'Integration Architecture è l'insieme dei protocolli che caratterizzano la piattaforma e consentono di abilitare effettivamente lo scambio dei dati. Gli elementi essenziali dell'architettura sono:

- Un layer di autenticazione e autorizzazione: i proprietari degli asset devono poter stabilire chi ha i privilegi per accedere ai dati. Inoltre, lo stesso NDT deve poter autenticare gli utenti, i provider e le organizza-

zioni riconosciute all'interno dell'ecosistema. Quindi, è necessario prima di tutto un meccanismo di autenticazione che riconosca gli utenti legittimi del sistema, dopodiché anche un sistema di autorizzazione che gestisca i permessi delle varie parti.

- Un repository contenente i riferimenti di tutti i Digital Twins dell'ecosistema: in un sistema distribuito di così grandi dimensioni è di fondamentale importanza avere un catalogo o più in generale un repository in cui mantenere tutti i riferimenti dei Digital Twins che popolano l'ambiente.

- Un protocollo per poter individuare i Digital Twins: un repository senza la capacità di richiedere un singolo elemento sarebbe inutile. Quindi nasce spontaneamente l'esigenza di un protocollo di discovery che consenta di ottenere la locazione dei Digital Twins e di utilizzarli in copia o in riferimento.

- Un linguaggio e un protocollo di query: una volta ottenuti i Digital Twins è necessario poter almeno effettuare query sullo stato presente e passato al fine di condurre una qualsiasi tipologia di analisi. Sfruttando l'ambiente distribuito saranno possibili anche query molto più complesse come descritto precedentemente.

- Meccanismi per trasformare e validare i dati: alcuni Twin Builder o alcune organizzazioni potrebbero non esporre direttamente i propri servizi utilizzando FDM ed RDL, perciò si potrebbe pensare di creare opportuni engine che aiutino l'IMF a trasformare i dati in un formato adeguato per la fruizione dei servizi. Inoltre, in parallelo deve sempre essere presente un engine che verifichi la validità degli scambi di dati al fine di assicurare la consistenza generale del framework.

In [13] vengono presi in considerazione gli elementi essenziali descritti precedentemente e vengono proposti sotto forma di componenti dell'Integration Architecture posizionandoli a diversi livelli a seconda del pattern scelto (pattern descritti successivamente):

- Layer di autenticazione e autorizzazione: tenendo in considerazione la natura distribuita dell'ecosistema e soprattutto la copertura a livello nazionale necessaria, utilizzare un unico sistema di autenticazione centralizzato non è la scelta ideale. Visto che a livello nazionale sono già presenti soluzioni ad-hoc per fornire autenticazione, i quali utilizzano sistemi avanzati per poter riconoscere ed assicurare l'identità dell'utente, l'NDT ha deciso di fare affidamento su questi, liberando il framework da un componente centrale di autenticazione. Possono essere utilizzati molteplici provider come: Post Office, Experion, Microsoft, Amazon, Google e OneLogin al fine di autenticare l'utente al sistema.
  Una volta autenticato, l'utente deve essere autorizzato ad usufruire dei vari servizi o dati a disposizione. Infatti, all'interno dell'NDT ogni utente ha specifici permessi e avrà la vista unicamente su ciò che potrà fruire. A tal fine è necessario un layer di autorizzazione. Questo verrà gestito internamente al framework attraverso un approccio basato su attributi. Sia i dati che gli utenti sono caratterizzati da un insieme di attributi che rappresentano i loro permessi. Se gli attributi dell'utente sono concordi con gli attributi richiesti dal dataset allora l'accesso viene consentito, altrimenti rifiutato.
  Quindi, in accordo con il principio *zero trust*[3], su cui si fonda il layer, in prima istanza viene autenticato l'utente, dopodiché viene autorizzato confrontadone gli attributi con quelli richiesti dal servizio o dal dataset a cui cerca di accedere e, in caso, viene fornito l'accesso.

- Reference Data Service: permette di eseguire query sull'insieme di dati che seguono l'RDL. È un componente o servizio che può trovarsi a livello locale o al core dell'architettura. Gli RDS che si trovano agli edge della rete dovranno sincronizzarsi con il core RDS attraverso un pattern pubblish/subscribe tramite il quale vengono forniti gli eventi di aggiornamento. La parte locale di questo componente serve per fornire un sistema

[3] `https://en.wikipedia.org/wiki/Zero_trust_security_model`

di caching che mantenga in locale tutte le informazioni più importanti e più frequentemente richieste all'interno dell'organizzazione.

- Catologo NDT: come descritto precedentemente, è necessario un catalogo contenente i riferimenti ai Digital Twins presenti in tutto l'ecosistema sopratutto ai fini della "discoverability". I cataloghi NDT possono essere strutturati in forma di "cataloghi di cataloghi" nei quali invece di mantenere il riferimento direttamente dei Digital Twins, viene mantenuto il riferimento ad altri cataloghi NDT.
  Inoltre, solitamente vi saranno pochi Cataloghi vicini al core dell'architettura, perciò è necessario un componente che fornisca un sistema di caching per gli edge, questo viene chiamato Local Directory Manager.

- Local Directory Manager: ha la funzione di cache locale per il catalogo NDT. Mantiene i riferimenti dei Digital Twins più importanti a livello dell'organizzazione in cui opera. Ovviamente, deve mantenere un link con uno dei cataloghi al core dell'architettura al fine di mantenere una visione consistente sull'ecosistema.

- Information management service: per poter utilizzare tutti i servizi di cui l'IMF si compone è necessario fornire un entry-point. L'information management service serve proprio a fornire l'interfaccia per utilizzare qualsiasi servizio del framework: discovery, query e push di dati. È costruito seguendo il principio dell'incapsulamento: i servizi veri e propri vengono tenuti nascosti dall'esterno, fornendo solamente un'interfaccia di utilizzo attraverso un layer di microservizi. In questo modo si rende possibile e molto più agevole l'aggiornamento delle componenti interne in quanto gli attori esterni non avranno dipendenze dagli elementi dell'architettura, ma solo dall'interfaccia a microservizi stessa.

- Log / Management (Auditing): un sistema distribuito a livello nazionale deve avere una componente che effettui un log continuo di tutti gli eventi interni ed esterni all'architettura. In questo modo in caso di errori sarà

molto più semplice effettuare un'analisi per scoprire l'origine del problema.

Lo stesso componente analizzando i log e le performance del framework deve poter comandare lo scaling al fine di adattarsi alle esigenze degli utenti.

Inoltre, vengono proposte anche 3 architetture di base: centralizzata, federata, distribuita.

L'architettura centralizzata (Figura 2.2) prevede la maggior parte dei componenti al core dell'architettura e quindi veramente pochi servizi (solamente quelli per il caching eventualmente) agli edge. Tutti gli utenti e le organizzazioni interagiscono con lo stesso nodo centrale. Questo porta a ovvi vantaggi dal punto di vista della gestione delle informazioni; infatti, trovandosi in un unico punto è molto più semplice assicurare la consistenza e il coordinamento dei riferimenti. Uno degli svantaggi però è dato dalla presenza di un single point of failure che può rappresentare un problema di notevole importanza in quanto per i principi di Trust e Function (Gemini Principles [11]) l'IMF deve essere sempre disponibile ed operativo.

L'architettura federata prevede più sistemi indipendenti ed interoperabili che comunicano tra loro attraverso un nodo centrale minimale. Il nodo centrale contiene le funzionalità di maggiore importanza che non possono essere delegate a terzi, come ad esempio la gestione delle autorizzazioni. Mentre, per quanto riguarda gli altri servizi essi vengono resi disponibili da provider esterni (autenticati) che sono in linea con i principi e gli standard dell'NDT.
Il punto di forza di questa architettura è che, oltre alle entità governative, anche le terze parti come banche e aziende possono fornire i servizi di cui l'Integration Architecture si compone, riducendo la mole di lavoro per la gestione del core, al costo di una gestione della consistenza e delle informazioni molto più complessa rispetto all'architettura centralizzata.

L'architettura distribuita spinge ancora di più le funzioni all'interno dei nodi all'edge dell'infrastruttura, lasciando il core con pochissime funzionalità. Caratteristica di questa architettura è la forte duplicazione di servizi sia a

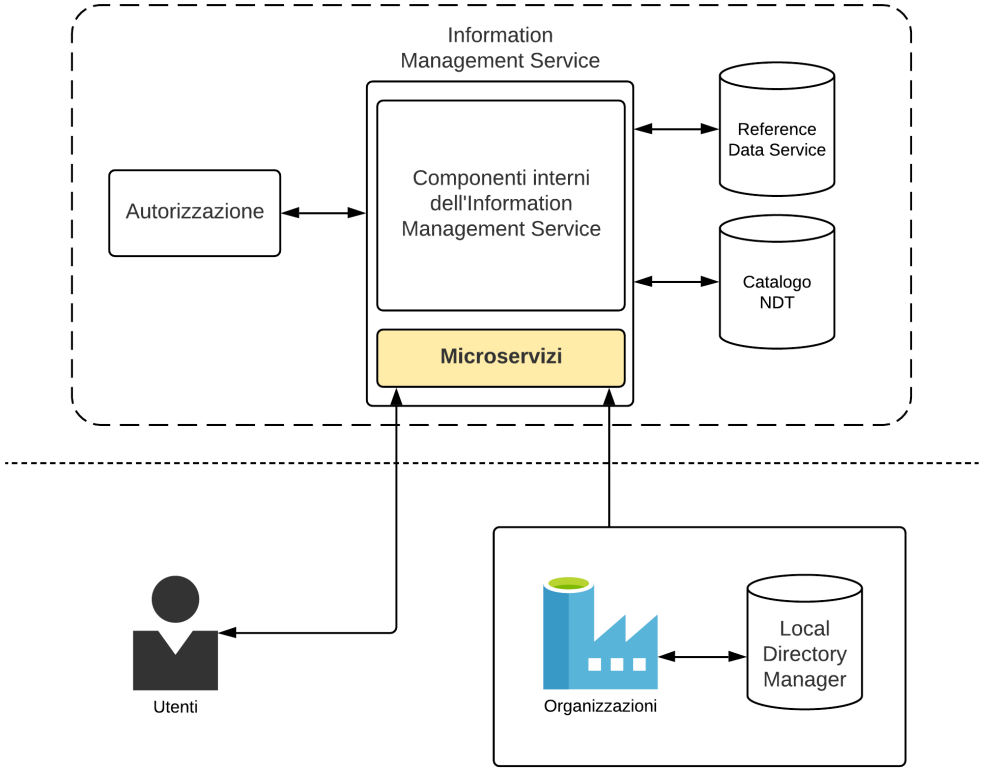

Figura 2.2: Proposta di Architettura centralizzata

livello locale che a livello di core.

Il CDBB e il Task Group propongono un ulteriore architettura di riferimento che prende in considerazione elementi di quella centralizzata, federata e distribuita al fine di creare un'architettura scalabile e componibile.

L'architettura di riferimento prevede un core ispirato alla proposta centralizzata in modo da fornire un unico punto in grado di assicurare la consistenza generale all'interno dell'ecosistema. Dopodiché, vengono creati ulteriori nodi, collegati al core, che forniscono l'interfaccia per le organizzazioni o per gli utenti finali. Essi replicano alcuni componenti dell'architettura al fine di aumentare le prestazioni e la scalabilità: Local Directory Manager, Reference Data Service e un layer di autorizzazione che, assieme, consentono di memo-

rizzare i riferimenti, i dati e i permessi dei Digital Twins d'interesse per le organizzazioni collegate.

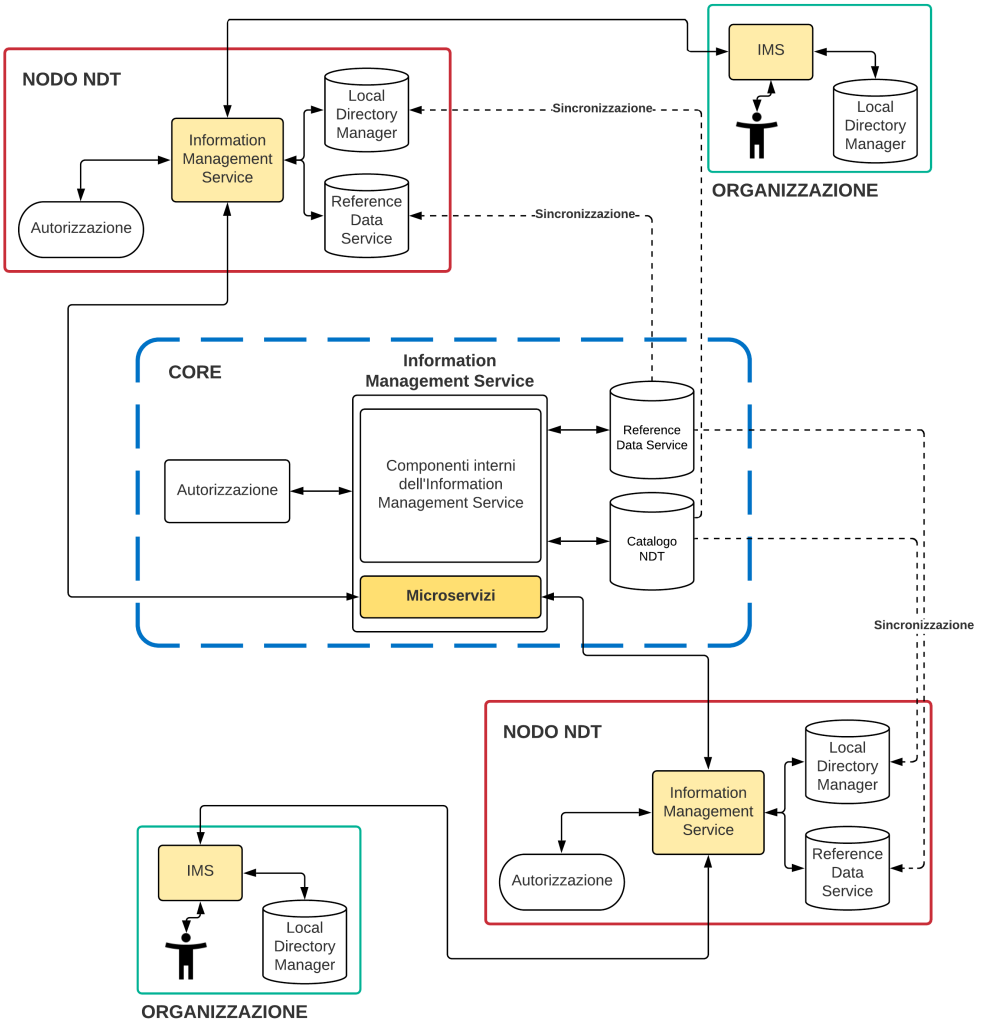

Figura 2.3: Architettura raccomandata

La creazione dei nodi intermedi al quale si connettono gli attori del National Digital Twin è di fondamentale importanza per le prestazioni generali del sistema e sopratutto per la scalabilità. Infatti, ogni singolo nodo, assieme al core, può scalare indipendentemente assicurando una grande resilienza in numerosi casi d'uso.

Come si può notare dalla Figura 2.3, i componenti locali mantengono un link con quelli del core per effettuare una sincronizzazione continua e mantenere una visione aggiornata e consistente dell'ecosistema.

Inoltre, tutti i flow che partono da un utente o da un'organizzazione seguono i link dell'architettura passando per il core.
In caso di query, l'utente dell'organizzazione comunica con l'interfaccia a microservizi del proprio IMS locale (Information Management Service), il quale spedisce la richiesta all'IMS del nodo NDT a cui è collegato. Se il dato richiesto è contenuto all'interno del Local Directory Manager e del Reference Data Service del nodo allora verrà restituita la versione in cache, altrimenti la query verrà spedita al Core in modo da ottenere i dati aggiornati. Nel secondo caso, la cache del nodo NDT intermedio verrà aggiornata con i nuovi dati.
Un altro impiego del framework è la pubblicazione di nuovi dati all'interno dell'ecosistema. In questo caso, il Data owner spedisce le informazioni dal proprio IMS locale all'IMS del nodo NDT a cui si è collegati (in modo da renderli disponibili all'interno dell'organizzazione d'interesse), dopodiché viene notificato il Core che, tramite pattern pubblish/subscribe, spedisce l'evento di aggiornamento a tutti i nodi NDT interessati.

I nodi all'interno del NDT possono essere sviluppati da qualsiasi organizzazione, associazione, provider di servizi che sia regolarmente autenticato allo stesso NDT. Non è importante quali strumenti si utilizzino per implementare il nodo. Infatti, possono essere utilizzati software open source, servizi cloud e così via. Però, è necessario implementare l'architettura proposta ed utilizzare gli standard definiti al fine di assicurare l'interoperabilità.

Il framework è tutt'ora in corso di sviluppo.

2.3 Proposte provenienti dalla ricerca

La ricerca offre molti spunti per la creazione di un'architettura moderna a supporto dei Digital Twins. In [6] viene descritta a livello concettuale una possibile architettura per l'impiego dei Digital Twins nel settore manifatturiero.

Questa visione, essendo concentrata sul settore manifatturiero, presta molta attenzione agli aspetti di simulazione, analisi e prevenzione i quali consentono di ottenere un notevole risparmio di denaro nelle fasi di produzione e operazione.

L'architettura è formata da diversi manager: Virtualisation Manager, Monitoring Manager, Decision-Making Manager, Simulation Manager e l'Interoperability Manager come illustrato nella Figura 2.4.

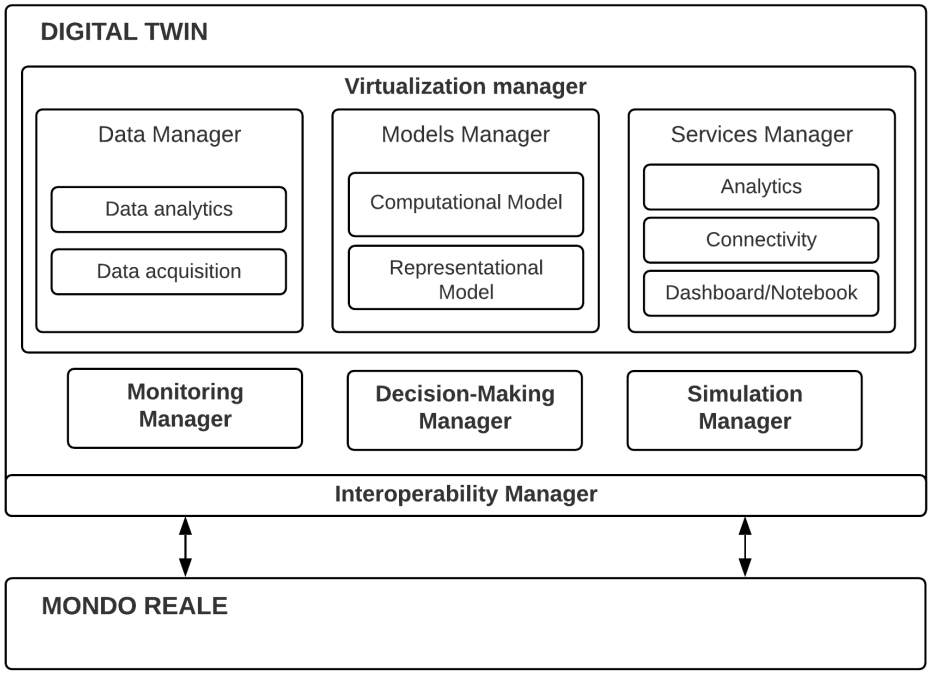

Figura 2.4: Schema architetturale

Il Virtualization Manager è formato a sua volta da tre componenti:

- Data Manager: si occupa di tutta la parte di acquisizione e analisi dati.
- Models Manager: si occupa della modellazione della replica virtuale con l'obiettivo di rappresentare nel modo più fedele possibile la controparte fisica. Include una parte di modellazione "rappresentativa" che punta

a descrivere proprietà ed attributi del Digital Twin con la possibilità di sfruttare ontologie esistenti e una parte "computazionale" utile per la conduzione di analisi, simulazioni e procedure al fine di esprimere o predire i cambiamenti dell'asset all'interno del contesto analizzato.

- Services Manager: al fine di rendere disponibili i dati all'esterno è necessario un layer di servizi che sia scalabile per poter soddisfare un ampio numero di richieste. Il layer potrebbe cambiare o adattare la propria interfaccia a seconda del caso d'uso dell'agente o in generale dell'attore che si collega al Digital Twin. Infatti, a seconda del settore d'interesse, si potrebbe offrire una vista diversa sul Digital Twin adatta al suo contesto.

Il Monitoring Manager ha la funzione di monitorare la connettività tra l'asset fisico e la parte di virtualizzazione della replica fornita dal Virtualizazion Manager.

Il Decision-Making Manager converte i feedback provenienti dai risultati dei processi di simulazione e analisi preventiva in un formato adatto ad essere fruito dagli attori esterni.

Infine, il Simulation Manager, come dice il nome, fornisce i meccanismi abilitanti per la simulazione.

Un ulteriore aspetto interessante affrontato in questo paper è l'approccio a microservizi nello sviluppo di ogni componente dell'architettura. Le architetture a microservizi offrono notevoli vantaggi. Innanzitutto, il codice può essere modificato molto più facilmente in quanto si può accedere direttamente alla porzione di codice interessata senza toccare la restante parte dell'applicazione. Dopodiché i vari servizi possono essere sviluppati da team diversi utilizzando stack e linguaggi di programmazione completamente diversi. Infine, in questo modo, i componenti dell'architettura possono scalare indipendentemente, andando di fatto a ridurre il costo adattandosi ai load delle varie componenti.

Questa architettura offre notevoli spunti e nuovi punti di vista che possono essere adottati nello sviluppo di un moderno middleware per Digital Twin. Infatti, la possibilità di modificare l'interfaccia o il modello esposto a seconda

del caso d'uso dell'attore che interagisce con il Services Manager e l'approccio a microservizi rappresentano concetti importanti che sicuramente verranno presi in considerazione nel corso di questa tesi.

In aggiunta, in [17] è offerta una proposta di architettura event-driven basata su una visione molto più aperta, volta all'interoperabilità, del concetto di Digital Twin. Come si potrà osservare nel capitolo seguente, le soluzioni esistenti presentano una componente di elevata verticalità nel settore di interesse che le rendono difficilmente adottabili per applicazioni che lavorano inter organizzazione. Una visione più aperta del concetto vede, come il National Digital Twin programme, la presenza di un ecosistema di Digital Twins connessi, interoperabili e pervasivi. Ogni Digital Twin deve essere progettato per poter ricevere i dati da più sorgenti, per essere osservabile e rispondere a richieste che sfruttano protocolli e tecnologie diversificate. Inoltre, la flessibilità rappresenta un altro fattore chiave. Nei casi in cui i requisiti temporali per effettuare lo shadowing siano molto stringenti si potrebbe prevedere un'implementazione dell'architettura che si trova direttamente all'edge, accompagnata da un'implementazione in cloud che presenta meno proprietà, ad esempio lo stato generale del sistema, con requisiti temporali molto più rilassati. Grazie a questa flessibilità, si riuscirebbero a modellare anche situazioni complicate da gestire interamente in cloud, come ad esempio operazioni particolari nel settore dell'healthcare.

L'architettura proposta in questo paper è event-driven, cioè guidata dagli eventi. Gli eventi che possiamo trovare all'interno della soluzione possono essere scatenati dall'asset fisico (e_{PA}), dal Digital Twin (e_{DT}) o direttamente dall'architettura (e_i). I primi dipendono da un cambiamento di stato dell'asset fisico o da un comando impartitogli e comunicato attraverso l'interfaccia presente tra l'asset e la sua replica. I secondi dipendono dall'evoluzione della replica e dal suo ciclo di vita. Infine, gli eventi interni sono prodotti dall'architettura a fronte delle normali attività ed azioni che fanno parte dei suoi compiti. Questi eventi, sopratutto quelli del Digital Twin devono poter essere osservati da attori esterni in modo da poter condurre analisi e simulazioni sopra ad essi.

I componenti di questa architettura sono semplici, però introducono alcuni concetti importati (Figura 2.5):

- Physical Interface: si occupa della comunicazione continua con l'asset fisico a partire dal binding e per tutto il ciclo di vita con il processo di shadowing. L'interfaccia può essere composta da diversi adapter al fine di assicurare l'interoperabilità e la possibilità di acquisire dati da sorgenti diverse. In questo modo possiamo avere un Digital Twin che riceve i dati in input, ad esempio, da un broker Mqtt e contemporaneamente da un altro adapter CoAP raccogliendo i dati da tutte le sorgenti necessarie (che quindi possono essere eterogenee).

- Digital Interface: è la componente che si occupa di rendere disponibili gli eventi e i dati del Digital Twin agli attori esterni. Anche qui, per assicurare l'interoperabilità, si adotta l'approccio ad adapter. In questo modo possiamo fornire i dati ad un'applicazione tramite Mqtt e contemporaneamente anche ad un altro agente tramite CoAP o tramite APIs.

- Digital Twin's Model Engine: è l'engine che gestisce tutto il comportamento del Digital Twin e che reagisce agli eventi interni andando ad orchestrare la gestione dei dati e delle informazioni. È l'engine che a fronte di un evento interno spedisce i dati ai componenti di Cache e Storage e a quello di Logging.

- Event-driven Engine: è il componente che gestisce il flusso dei dati, in particolare gestisce gli eventi e il loro flow all'interno dell'architettura, connettendo assieme tutti i vari componenti.

- Cache e Storage: gestisce i dati associati al Digital Twin.

- Logging e Monitoring: effettua il log di tutti gli eventi e monitora il sistema.

Anche questa architettura, come la precedente, offre alcuni punti di studio da prendere in considerazione. Primo su tutti la possibilità di avere diversi

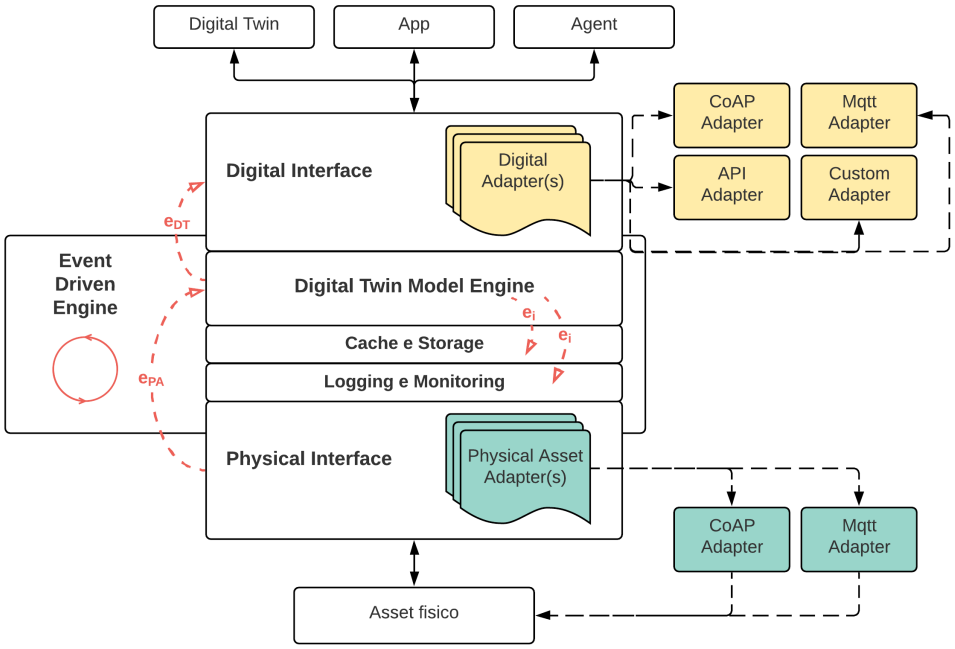

Figura 2.5: Schema architetturale

adapter sia a livello di interfaccia con il Digital Twin, sia a livello di interfaccia con gli attori esterni. Inoltre, l'approccio event-driven si presta molto bene all'utilizzo in architetture a microservizi come quelle viste precedentemente.

2.4 Riepilogo

Queste architetture sono state pensate per essere il più possibile agnostiche dalle particolari tecnologie ed alcune di esse hanno, a mio parere, l'intenzione di rimanere solamente delle linee guida architetturali al fine di guidare la progettazione e lo sviluppo di piattaforme "compliant".

Le proposte qui descritte rappresentano una visione più aperta ed interoperabile del concetto di Digital Twin che non si limita a rappresentare repliche digitali all'interno di un contesto estremamente verticale, ma fa i primi passi

verso la creazione di ecosistemi in cui i Digital Twins possono appartenere a contesti diversi o almeno verso la creazione di piattaforme in grado di trattare Digital Twins con modellazioni che spaziano su più settori differenti.

Capitolo 3

Piattaforme e frameworks disponibili

Le soluzioni precedenti non forniscono, nella maggior parte dei casi, nessuna specifica implementativa e sono indipendenti da particolari tecnologie, pattern di sviluppo o linguaggi di programmazione. Perciò, lasciano ampio spazio per la descrizione di quello che, ad oggi, esiste sul mercato e viene utilizzato nei primi progetti, aziendali e non, che adottano il concetto di Digital Twins all'interno dei propri processi.

In questo capitolo verranno illustrate le principali soluzioni presenti sul mercato ed utilizzabili al fine di implementare i propri sistemi o addirittura al fine di realizzare una delle architetture proposte nel capitolo precedente. Dopo aver fatto una panoramica, mi soffermerò su Eclipse Ditto e in particolare sulla soluzione di Microsoft Azure in quanto è stata soggetta di un mio studio approfondito.

3.1 Panoramica

Il mercato, dal 2002 ad oggi, non è rimasto un attore secondario, anzi, offre tantissime soluzioni che si approcciano al mondo dei Digital Twins in modo diversificato, principalmente seguendo lo schema e la direzione già presente a

livello aziendale. Ora andrò a fare una breve panoramica delle soluzioni più famose ad oggi disponibili con qualche caratteristica che li contraddistingue:

- Eclipse Ditto: è la soluzione open source creata dal team di Eclipse che incapsula l'approccio a "Digital Twins" all'interno dello sviluppo di progetti IoT. È una delle soluzioni più versatili dopo Microsoft Azure Digital Twins. `https://www.eclipse.org/ditto/index.html`

- Microsoft Azure Digital Twins: è un servizio cloud di tipo PaaS (Platform as a service) che permette la creazione di grafi di conoscenza basati sui modelli digitali di interi ambienti ad esempio: palazzi, fattorie, fabbriche, treni, stadi, ospedali o intere città. `https://docs.microsoft.com/en-us/azure/digital-twins/overview`

- XM Pro: è una piattaforma "no code" per lo sviluppo e il deploy di soluzioni basate su Digital Twin. Presenta un sistema chiamato "Event Intelligence", che osserva gli eventi presenti all'interno delle serie temporali real-time, applica le analisi desiderate ed esegue predizioni al fine di fornire azioni raccomandate. Inoltre, i Digital Twins possono ricevere dati da più sorgenti di dati le quali vengono integrate automaticamente all'interno della piattaforma. Ad esempio, un Digital Twin potrebbe ricevere dati da un sistema ERP, dai sensori presenti sull'asset e dal sistema di gestione PLM. `https://xmpro.com/`

- Bentley iTwin: è una piattaforma cloud aperta e scalabile che fornisce API e servizi per aiutare gli sviluppatori nella progettazione e nella realizzazione di applicazioni basate su Digital Twin per la creazione, visualizzazione e l'analisi delle replice digitali dei propri asset fisici. Viene utilizzato spesso in collaborazione con i prodotti Microsoft, soprattutto Azure Digital Twins. `https://www.bentley.com/en/products/product-line/digital-twins`

- General Eletric: l'azienda ha sviluppato diversi servizi volti al settore industriale. Un modo per descrivere gli obiettivi del sistema è "See - Think

- Do". Fornisce diverse funzionalità: gestione della connessione con gli asset, algoritmi di analisi, Machine Learning e sistemi per il monitoraggio delle performance del prodotto. Il sistema di Digital Twins è basato sulla piattaforma Predix[1] mettendo in evidenza la spinta di questo sistema verso l'analisi e la predizione di fenomeni.

- PTC Windchill: ha sviluppato un sistema PLM in accordo con la prima versione di Digital Twin offerta da Grieves.

- Dassault Systèmes: si è inserito nel settore manifatturiero per il controllo e la validazione delle operazioni in essere.

- DXC: in collaborazione con Microsoft, ha sviluppato un sistema basato su Digital Twins per la previsione delle performance nelle macchine ibride.

- Siemeens Simcenter: è un sistema sviluppato da Siemeens per la simulazione di sistemi attraverso modellazioni 3D e test per la previsione delle prestazioni in situazioni e ambienti critici. Questo sistema accompagna il ciclo di vita dei prodotti dalla loro creazione fino alle fasi finali consentendo uno sviluppo affidabile in tempi più rapidi. `https://www.plm.automation.siemens.com/global/it/products/simcenter/`

- Bosh IoT: è una soluzione che integra la gestione dell'infrastruttura IoT con la possibilità di adottare l'approccio a Digital Twin. Utilizza internamente il framework Eclipse Ditto. `https://bosch-iot-suite.com/`

- Autodesk Digital Twins: è la soluzione proposta da Autodesk per utilizzare i vantaggi dei Digitial Twin all'interno del settore delle costruzioni e dell'architettura in generale. `https://www.autodesk.com/solutions/digital-twin/architecture-engineering-construction`

Durante il periodo di tesi mi sono concentrato sull'approfondimento delle due soluzioni principali presenti ad oggi sul mercato: *Eclipse Ditto* e in parti-

[1] `http://www.predix.com`

colare *Microsoft Azure Digital Twins* in quanto, soprattutto il secondo, sono tra gli approcci più versatili disponibili ed utilizzati.

3.2 Eclipse Ditto

Con Eclipse Ditto qualsiasi asset può essere utilizzato come un servizio web attraverso il suo Digital Twin. Ditto, non è una piattaforma IoT completa. Infatti, non prevede l'implementazione di alcun protollo IoT al fine di comunicare con i device. Il suo scopo è unicamente quello di fornire APIs per le applicazioni web, mobile o altri servizi di backend che utilizzano WebSocket, AMQP oppure HTTP al fine di accedere all'ultimo stato conosciuto di un determinato asset rappresentato attraverso il suo Digital Twin. L'accesso può essere ristretto a solo gli utenti o i servizi autorizzati. Ultimamente è stato aggiunto anche un Broker MQTT 3.1.1.
Le entità principali del modello sono:

- Thing: sono gli elementi modellati come Digital Twin, quindi qualsiasi tipologia di asset fisico o device virtuale. Ogni Thing possiede il proprio ThingID e fa riferimento ad una policy in modo da definire i permessi su quel Thing. Un Thing può contenere una Definition la quale descrive le capacità e le features del Digital Twin. La Definition viene spesso modellata attraverso Eclipse Vorto[2]. Dopodiché, è possibile inserire attributi, di qualsiasi tipo, per descrivere un Thing. Gli attributi vengono solitamente utilizzati per definire i tipi di dati statici, cioè quelli con una bassa (quasi nulla) frequenza di aggiornamento. Invece, se vogliamo descrivere dati che cambiano frequentemente allora utilizziamo le features. Un Thing può contenere un numero arbitrario di features. Inoltre, gli attributi e le features possono essere descritti da metadati.
  Di seguito un esempio della definizione di un Thing prelevato dalla documentazione ufficiale.

```
{
```

[2] `https://www.eclipse.org/vorto/`

```
  "thingId": "the.namespace:theId",
  "policyId": "the.namespace:thePolicyId",
  "definition": "org.eclipse.ditto:HeatingDevice:2.1.0",
  "attributes": {
      "someAttr": 32,
      "manufacturer": "ACME corp"
  },
  "features": {
      "heating-no1": {
          "properties": {
              "connected": true,
              "complexProperty": {
                  "street": "my street",
                  "house no": 42
              }
          },
          "desiredProperties": {
              "connected": false
          }
      },
      "switchable": {
          "definition": [ "org.eclipse.ditto:Switcher:1.0.0" ],
          "properties": {
              "on": true,
              "lastToggled": "2017-11-15T18:21Z"
          }
      }
  }
}
```

Listato 3.1: Esempio di un Thing prelevato da: `https://www.eclipse.org/ditto/basic-thing.html#example`

- Features: è un insieme di proprietà. Le proprietà, nel loro insieme, sono

rappresentate come un oggetto JSON. Ogni singola proprietà, a sua volta, può essere un semplice valore scalare oppure un oggetto JSON. Oltre che al valore attuale della feature, è possibile impostare anche un eventuale valore desiderato, utile in diversi contesti applicativi. Ogni feature può avere a sua volta una definizione, la quale può essere descritta tramite *Eclipse Vorto*.

- Policy: consente di configurare le autorizzazioni per l'accesso ai dati dei Digital Twins a diversi livelli di granularità.

- Thing Metadata: sono i metadati che descrivono le features e gli attributi. È possibile specificare metadati aggiuntivi da memorizzare rispetto ad una singola proprietà o un singolo attributo appartenente al modello di un Digital Twin. Un esempio tipico di metadato è la data e l'ora di ultimo aggiornamento.

- Autenticazione: utile al fine di verificare l'identità degli attori che vogliono accedere al servizio.

- Autorizzazione: utile per verificare se l'utente che si appresta ad accedere ad un Digital Twin dispone di tutti i permessi necessari.

Eclipse Ditto mette anche a disposizione la possibilità di spedire messaggi attraverso un canale (o link) bidirezionale tra l'asset e il Digital Twin. Non si tratta di un comando a cui il framework Ditto risponde, bensì un semplice messaggio contenente un payload da interpretare.
È possibile spedire un messaggio da un Digital Twin ad un device per eseguire un'azione su di esso, oppure è possibile spedire un messaggio al Digital Twin, che verrà poi interpretato esternamente ad Eclipse Ditto.
Ovviamente, così come per ogni altra azione sui Digital Twin, sono necessari i giusti permessi per poter utilizzare i messaggi, in particolare: quelli di lettura per la ricezione e quelli di scrittura per l'invio.

Come si può notare ad Eclipse Ditto mancano notevoli funzionalità per poter implementare tutte le caratteristiche dei Digital Twins illustrate nel primo

capitolo. Però la sua struttura altamente modulare ed integrabile consente di poter collegare il framework a tantissimi altri servizi. Ad esempio, per poter gestire la comunicazione con gli asset fisici, quindi per tutta la parte di protocolli IoT e per i canali bidirezionali, si può utilizzare Eclipse Hono[3]. Inoltre, come visto, Ditto salva solamente lo stato attuale di un Digital Twin, perciò per poter avere la storia del Digital Twin al fine di condurre analisi e applicare algoritmi di Machine Learning è necessario collegarlo ad altri servizi di storage ed analisi come ad esempio InfluxDB[4] e Grafana[5] oppure lo stack ELK[6]: Elasticsearch - Log Search - Kibana oltre a tutti i servizi di analisi come quelli offerti da Microsoft o Apache.

Inoltre, internamente alla visione di Digital Twin implementata in Eclipse Ditto mancano i concetti di *relazione tra Digital Twins* e di *ereditarietà del modello* i quali sono molto utili ai fini della modellazione di problemi reali.

3.3 Azure Digital Twins

Una delle soluzioni più versatili e soprattutto più complete è quella offerta da Microsoft. Microsoft con Azure Digital Twins offre, come anticipato, un servizio cloud di tipo PaaS (Platform as a service) che permette la creazione di grafi di conoscenza basati sui modelli digitali di interi ambienti. Alcuni vantaggi rappresentativi di questo servizio sono:

- Possibilità di modellare qualsiasi ambiente e creare Digital Twin in un sistema già scalabile e sicuro

- Connettere gli asset fisici (ad esempio dispositivi Iot) ai Digital Twins e quindi permettere l'entanglement. Oppure connettere interi sistemi preesistenti ed arricchirli con le potenzialità dei Digital Twins.

[3] `https://www.eclipse.org/hono/`
[4] `https://www.influxdata.com/`
[5] `https://grafana.com/`
[6] `https://www.elastic.co/`

- Gestione della dynamic business logic e del data processing attraverso un approccio *event-driven*.

- Possibile integrazione con gli altri servizi di Azure come Azure Time Series Insights, AI, Analytics per poter utilizzare la storia del Digital Twin e prevedere il futuro.

In Azure Digital Twins, le entità logiche (quindi i Digital Twins), che rappresentano gli asset, vengono descritte utilizzando modelli. I modelli sono definiti in un linguaggio JSON-like chiamato *Digital Twins Definition Language* o *DTDL* e descrivono il Digital Twin in termini delle loro proprietà, dei dati telemetrici, degli eventi, dei componenti e delle relazioni. I modelli possono definire relazioni (*relationships*) tra le varie entità in modo da connettere i vari Digital Twins creando un grafo di conoscenza che rifletta le varie interazioni.

I modelli dei Digital Twins descritti in DTDL, una volta instanziati (si può pensare all'analogia tra una classe e l'oggetto che ne è istanza), sono rappresentazioni aggiornate e vive del mondo reale. Utilizzando le relazioni, come detto precedentemente, all'interno dei modelli, descritti in DTDL, possiamo connettere i Digital Twins e creare un grafo che rappresenti il nostro ambiente. Microsoft mette a disposizione alcuni strumenti per vedere realmente il grafo come ad esempio: Azure Digital Twins explorer[7]. Al fine di ottenere i dati dal grafo di conoscenza presente in Azure Digital Twins è presente un potente sistema di query fornito tramite API. Esso consente di eseguire query su tutto il grafo ed ottenere proprietà, relazioni, proprietà delle relazioni, informazioni sul modello e tanto altro.
Inoltre, il servizio di Azure Digital Twins prevede un ricco sistema ad eventi che consente di gestire il data processing e la business logic. Possono essere connesse risorse computazionali esterne come ad esempio le Azure Functions per processare i dati in modo scalabile e personalizzato.

Azure Digital Twins ha in comune con Eclipse Ditto il fatto di essere un servizio dedicato ad offrire funzionalità strettamente legate al concetto di Digi-

[7] `https://docs.microsoft.com/it-it/azure/digital-twins/concepts-azure-digital-twins-explorer`

tal Twin. Azure Digital Twins gestisce solamente lo stato attuale della replica, quindi non modella la sua storia, né tanto meno si preoccupa della connettività con l'asset fisico. Inoltre, manca anche di qualsiasi componente di analisi (non considerando il sistema di query) o algoritmi di Machine Learning. Però, a differenza di Eclipse Ditto, nella soluzione Microsoft viene gestito in modo molto più agevole. Infatti, è un approccio completamente a microservizi costruito in modo tale che per implementare una soluzione completa basta utilizzare tutti gli altri servizi offerti in casa Microsoft. Questo assicura anche una forte scalabilità della soluzione, in quanto l'approccio a microservizi consente ad ogni componente di poter scalare indipendentemente.
Sono presenti servizi di qualsiasi tipo: gestione dispositivi IoT, storage di serie temporali, analisi, Machine Learning, codice serverless, message bus, gestori di eventi e altro ancora. L'architettura è completamente event-driven. Ogni servizio utilizzato produce un insieme di eventi di diverso tipo i quali comunicano all'esterno le informazioni necessarie. In questo modo, è possibile collegare numerosi servizi senza la necessità di essere legati ad un particolare produttore, ad esempio Microsoft, ma con la possibilità di scegliere per ogni funzionalità da implementare il servizio più adatto al contesto.

Al fine di connettere gli asset fisici ed aggiornare automaticamente l'ambiente creato e modellato su Azure Digital Twins è possibile utilizzare Azure Iot Hub. Esso permette di collegare in modo scalabile tantissimi asset fisici direttamente come Iot Devices e mette a disposizione vari strumenti, tra cui un canale bidirezionale tra l'asset e l'hub, aprendo la strada per l'applicazione del principio dell'Entanglement e in particolare dell'Association nella sua massima espressione. Questo consente di raccogliere i dati tramite Azure IoT Hub, spedirli ad Azure Digital Twins per aggiornare la replica digitale, eseguire analisi sui dati sfruttando ulteriori servizi Microsoft, prendere una decisione e spedire il comando direttamente all'asset fisico.

Invece, per quanto riguarda l'output, utilizzando l'event-system presente in Azure Digital Twins, è possibile spedire i dati a qualsiasi servizio di analisi o storage di serie temporali. Azure Digital Twins utilizza le *event routes* e gli *endpoints* al fine di spedire gli eventi e quindi i dati all'esterno della piattafor-

ma. Una event route consente di spedire i dati, associati agli eventi prodotti dai Digital Twins, a gli endpoint creati all'interno del gruppo risorse di Microsoft, consentendo di specificare anche alcune policy personalizzate (ad esempio per filtrare il tipo di eventi che devono raggiungere un determinato endpoint). Possono essere collegati tre tipi di endpoint, tutti appartenenti all'ecosistema Microsoft: Event Hub, Event Grid e Service Bus. Lo scopo degli endpoint è consentire la fruizione degli eventi e dei dati, quindi agire da sorgenti per i servizi dell'architettura al fine di consentire l'implementazione delle varie funzionalità.
Ad esempio, se fosse necessario elaborare i dati prodotti dai Digital Twins per la creazione di dati derivati potremmo collegare una Azure Function la quale offre la possibilità di eseguire codice serveless. Inoltre, è possibile spedire i dati ad ulteriori destinazioni, con la possibilità di implementare diversi pattern come il Pubblish/Subscribe o l'Observer (assicurando real time o near real time) tramite servizi come Azure Signal-R, message broker o message bus presenti in Microsoft o di terze parti. Se volessimo memorizzare i dati storici e le serie temporali si potrebbero utilizzare servizi come Azure Data Lake o Azure Time Series Insights per poi eseguire analisi e predizioni tramite ulteriori servizi come Azure Stream Analytics o Azure Synapse Analytics.

Questi sono solo alcuni servizi presenti all'interno dell'ecosistema Microsoft ma, come accennato precedentemente, grazie all'architettura profondamente a microservizi event-driven è possibile, sfruttando i vari endpoint, message bus e message broker assieme alle REST APIs e agli SDK, implementare le funzionalità mancanti in Azure Digital Twins utilizzando qualsiasi altro servizio compatibile. Quindi, ad esempio, non è necessario utilizzare Iot Hub per la connettività con l'asset fisico, ma appunto sfruttando le REST APIs e l'SDK possiamo guidare i Digital Twins da altre sorgenti come ad esempio Eclipse Hono o addirittura da un servizio sviluppato interamente ad-hoc.

Dunque, Azure Digital Twins in sé abilita la creazione e la modellazione dei Digital Twins attraverso il linguaggio DTDL, memorizza lo stato attuale delle repliche e le loro relazioni e quindi tutto il grafo di conoscenza, consente di eseguire query sul grafo attraverso la query API ed infine gestisce il potente

sistema ad eventi e le API per poter integrare e guidare la piattaforma con altri servizi o applicazioni client.

I Digital Twins, in Azure Digital Twins, vengono modellati utilizzando il *Digital Twins Definition Language (DTDL)*[8]. Questo linguaggio consente di specificare il vocabolario desiderato, implementare ontologie e di avere in generale una grande libertà nella modellazione. Un modello è simile al concetto di *classe* nella programmazione object-oriented.
Il DTDL è un linguaggio basato su JSON-LD in modo da essere indipendente dal linguaggio di programmazione adottato all'interno dei progetti. Il DTDL è utilizzato all'interno dell'ecosistema Microsoft anche per altri servizi, perciò presenta alcune features che in Azure Digital Twins non possono essere sfruttate, come i comandi.

All'interno di un modello DTDL, l'elemento al primo livello è sempre di tipo "Interface". Esso incapsula l'intero modello di un singolo tipo di Digital Twin. L'interfaccia, per poter essere descritta, può contenere i seguenti elementi:

- Property: esse vanno a formare lo stato di un Digital Twin. Sono come i campi all'interno di una classe in un linguaggio object-oriented. La piattaforma memorizza l'ultimo valore assunto dalle proprietà in modo tale da essere sempre disponibile per la lettura.

- Telemetry: i dati telemetrici rappresentano solitamente misurazioni o eventi. Infatti, questi non vengono memorizzati all'interno della piattaforma e quindi per poterli elaborare è necessario trattarli non appena si scatena l'evento di arrivo, altrimenti verranno eliminati subito. Quindi, sarà necessario rimanere in ascolto tramite un servizio esterno per poter effettuare il salvataggio o per effettuare una computazione.

- Component: vengono utilizzati per descrivere un componente che è parte integrante del Digital Twin, ma che non ha un'identità separata e quindi non necessita di essere creato e gestito indipendentemente. Vengono

[8] `https://github.com/Azure/opendigitaltwins-dtdl/blob/master/DTDL/v2/dtdlv2.md`

spesso utilizzati per organizzare al meglio il codice che contiene i modelli in quanto consente di riutilizzare elementi di modellazione, risparmiando codice e soprattutto mantenendo consistenza di modifica in quanto il componente è definito solo in un luogo.

- Relationship: sono il mezzo attraverso il quale è possibile creare effettivamente i legami che descrivono il grafo di conoscenza su cui si basa tutta la piattaforma. Infatti, le relationship consentono di rappresentare le relazioni tra Digital Twins. Ad esempio, *piano* "contiene" *stanza*, dove sia il piano che la stanza sono modellati come Digital Twin, oppure *macchina* "è di proprietà" di *persona*, dove macchina e persona sono di nuovo modellati come Digital Twin. Le relazioni possono avere a loro volta delle proprietà. Ogni singola istanza di relazione sarà, in questo caso, descritta da un insieme di proprietà.

Di seguito un esempio di modello definito tramite DTDL:

```
[
  {
    "@id": "dtmi:com:contoso:Planet;1",
    "@type": "Interface",
    "@context": "dtmi:dtdl:context;2",
    "displayName": "Planet",
    "contents": [
      {
          "@type": "Property",
          "name": "name",
          "schema": "string"
      },
      {
          "@type": "Property",
          "name": "mass",
          "schema": "double"
      },
      {
```

```
        "@type": ["Telemetry", "Temperature"],
        "name": "temp",
        "schema": "double",
        "unit": "degreeCelsius"
    },
    {
        "@type": "Component",
        "name": "deepestCrater",
        "schema": "dtmi:com:contoso:Crater;1"
    },
    {
        "@type": "Relationship",
        "name": "satellites",
        "target": "dtmi:com:contoso:Moon;1"
    }
  ]
},
{
  "@id": "dtmi:com:contoso:Crater;1",
  "@type": "Interface",
  "@context": "dtmi:dtdl:context;2",
  "contents" : [
      {
          "@type" : "Property",
          "name" : "componentProp",
          "schema" : "integer"
      }
  ]
},
{
  "@id": "dtmi:com:contoso:Moon;1",
  "@type": "Interface",
  "@context": "dtmi:dtdl:context;2"
}
```

```
]
```

Listato 3.2: Modello in DTDL di un Digital Twin di un pianeta

In questo esempio, si vuole modellare un semplice pianeta che riporta tre proprietà, rispettivamente nome, massa e temperatura, un componente che rappresenta il cratere più profondo e una relazione che serve per legare il pianeta al suo satellite. Come anticipato, il componente ci consente di definire un elemento di utilizzo comune all'interno del contesto rendendo uniche eventuali modifiche, mentre le relazioni ci consentono di creare gli archi del grafo di conoscenza. Nella prima parte del modello ci sono alcuni campi che sono presenti in tutti i modelli DTDL:

- @id: è l'identificatore del modello.
- @type: rappresenta il tipo di informazione che si sta descrivendo (Interface, Property, Telemetry...).
- @context: i modelli devono sempre indicare come contesto dtmi:dtdl:context;2
- displayName: opzionale, è un nome alternativo, più semplice da ricordare, per il modello.
- contents: contiene tutta la descrizione del modello, quindi la rappresentazione del Digital Twin. Corrisponde ad un array di definizioni in cui ogni elemento inserito deve essere descritto a sua volta dal @type (Property, Telemetry, Relationship o Component) per indicare il tipo di informazione rappresentata, dal name e dallo schema che indicano rispettivamente il nome e il tipo di dato memorizzato (double, string, ...)

Come si può notare, i modelli presentano un campo @id che identifica il modello in sé. I Digital Twins in Azure Digital Twins possiedono due livelli di identificatori. Il primo è appunto l'identificatore del modello, il quale viene specificato nella stesura del modello in DTDL e che identifica il modello come tale (in una visione semplificata, simile al nome di una classe in un linguaggio object-oriented). Il secondo è l'identificatore del Digital Twin che viene

specificato durante la sua creazione. Questo è come se fosse l'object identifier (OID) dell'oggetto creato a partire da una determinata classe in un linguaggio object-oriented. Infatti, nel momento della creazione oltre all'id del Digital Twin è necessario specificare anche l'id del modello a cui quel Digital Twin fa riferimento per la definizione della sua struttura. Questo significa che i Digital Twins presenti all'interno del grafo di conoscenza non sono altro che le istanze dei modelli specificati. A partire da un modello (che ha un proprio identificatore) è possibile creare n istanze ciascuna con una propria identità.

Inoltre, tornando all'esempio precedente si può notare che la proprietà "temp" è definita in modo diverso rispetto alle altre. Infatti, questa è una proprietà particolare a cui è stato applicato il *Semantic Type*. I Semantic Types consentono di esprimere un valore assieme alla sua unità di misura e possono essere utilizzati sia con le Property che con Telemetry. Questo consente di esprimere più informazioni di contesto utili per la descrizione della proprietà o del dato telemetrico, in aggiunta ai metadati presenti di default.

Un ulteriore feature molto utile offerta dal DTDL è l'ereditarietà. Grazie all'ereditarietà tra modelli è possibile creare un modello utilizzando come base un altro. Quindi, ad esempio, è possibile creare un modello base chiamato "Stanza" con la descrizione di base comune a tutti i modelli di stanze, dopodiché creare le versioni specializzate come "Palestra", "Conference Room" e così via. Questo consente di avere ancora più libertà e consistenza nella modellazione ed inoltre pone il linguaggio DTDL ad un livello più ingegneristico rispetto ad esempio alla modellazione presente in Eclipse Ditto.

Il nome *Azure Digital Twins* perciò, si riferisce al servizio offerto da Microsoft in sé. Il grafo di conoscenza (esempio in Figura 3.1) contenente i Digital Twins e le loro relazioni sono tutti elementi interni all'istanza del servizio Azure.

- API del piano di controllo: è un ARM (Azure Resource Manager) e quindi vengono utilizzate per gestire le risorse Azure (istanze di risorse e gruppi di risorse che sono alla base della gestione dei servizi Azure) come

ad esempio la creazione ed eliminazione dell'istanza del servizio Azure Digital Twins all'interno del proprio resource group.

- API del piano dati: utilizzate per le operazioni di gestione dei dati e dei Digital Twins, ad esempio creazione, modifica e eliminazione dei Digital Twins e dei rispettivi modelli, query ed operazioni sul grafo.

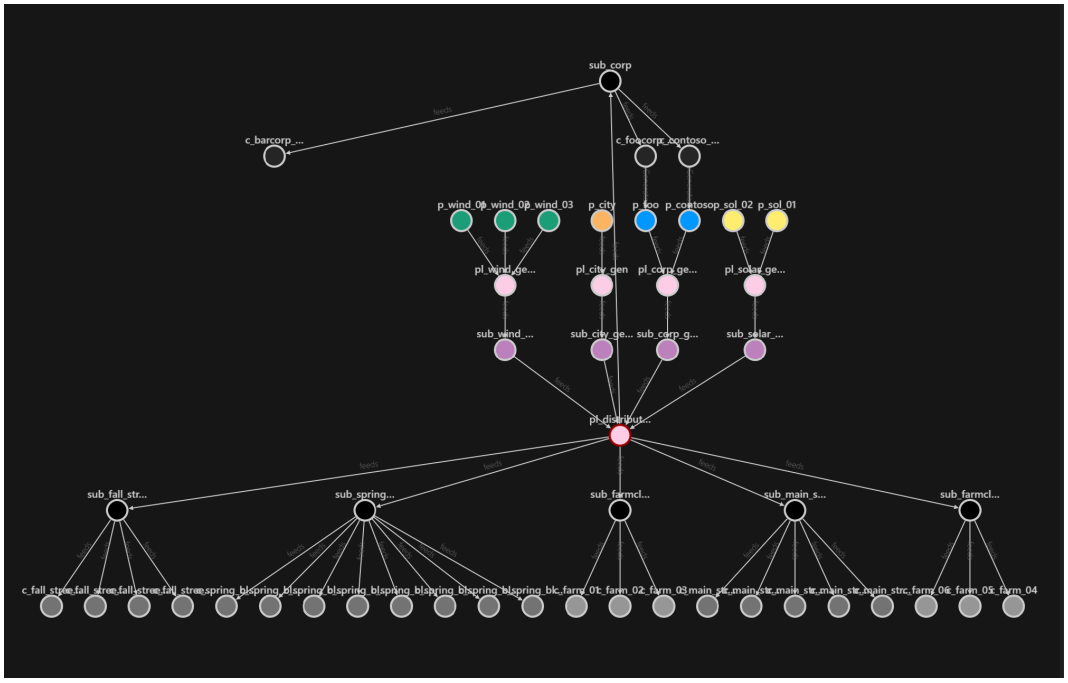

Figura 3.1: Esempio di grafo di conoscenza (prelevato dalla documentazione)

3.4 Riepilogo

In conclusione, le soluzioni presenti attualmente sul mercato presentano, nella maggior parte dei casi, una verticalità molto spinta verso il proprio settore aziendale.

Le soluzioni che attualmente sono rivolte alla visione più aperta ed interoperabile di Digital Twin sono solo a livello concettuale (come visto nel capitolo precedente), anche se, alcuni servizi, come quello di Microsoft potrebbero essere abilitanti per la realizzazione.

Infatti, Microsoft con la creazione del linguaggio DTDL e della piattaforma Azure Digital Twins ha spianato la strada per l'adozione dei Digital Twins in qualsiasi contesto e ambiente. Inoltre, Microsoft ultimamente sta parlando del concetto di *metaverse*[9] sottolineando, ancora di più, l'intenzione di creare uno strumento alla base della realizzazione di repliche, molto più estese ed interoperabili, della realtà.

[9] `https://mybuild.microsoft.com/sessions/f06287c8-8e56-452f-ae2f-e739c2be4870`

Capitolo 4

Dai Digital Twins a Web of Digital Twins

Dopo aver descritto l'evoluzione del concetto di Digital Twin nel capitolo 1 e le principali soluzioni in termini di architetture e framework a supporto nel capitolo 2 e nel capitolo 3, ora è necessario fornire la nuova visione, chiamata *Web of Digital Twins*, condivisa in questa tesi e fornita in [18], che prende in considerazione gli spunti descritti precedentemente per fornire nuovi punti di vista.

Web of Digital Twins rappresenta il riferimento principale per il progetto di questa tesi ed è un concetto ancora puramente di ricerca. L'obiettivo è cercare di dare una mia interpretazione e un mio contributo, se possibile, approcciandomi per la prima volta ad un argomento che è tutt'ora di ricerca.

4.1 Web of Digital Twins

Il problema già evidenziato dal progetto National Digital Twin è il *siloing* delle attuali soluzioni o visioni. Infatti, se continuassimo ad utilizzare interfacce e piattaforme proprietarie e closed-source, sarebbe veramente difficile creare ecosistemi di Digital Twins capaci di lavorare assieme ed essere interoperabili al fine di aumentare il valore delle informazioni offerte.

Analizzando anche alcuni concetti proposti prima dell'introduzione dei Digital Twins, si nota l'esigenza di poter modellare attraverso l'uso del software la realtà che ci circonda ottenendo una rappresentazione che possa essere aumentata ed utilizzata in modi fino a prima sconosciuti. Infatti, nei primi anni novanta, David Gelernter introdusse l'idea di *Mirror Worlds* [8] come *"software models of some chunk of reality"* e *"a true-to-life mirror image trapped inside a computer"*.

Basandosi fortemente su questa idea viene ideato in [18] il concetto di *Web of Digital Twins*. Esso ha l'obiettivo di scardinare la visione dominante nella ricerca e a livello aziendale che vede i Digital Twins come virtualizzazioni di asset fisici all'interno di un ambiente o di una piattaforma closed-source e difficilmente interoperabile. Vi è la necessità di creare un ecosistema di Digital Twins che possano comunicare tra organizzazioni diverse, assicurando un alto grado di interoperabilità.

Il Digital Twin, non essendo più strettamente legato solo ad un'organizzazione, viene utilizzato come un servizio grazie alla stratificazione dell'architettura proposta. Infatti, Web of Digital Twins definisce un layer cross-applicazione che consente il collegamento tra il mondo reale e quello digitale sopra al quale sono sviluppate le applicazioni che necessitano di interagire con gli asset fisici. In [18] si considera un approccio basato sugli agenti al fine di sviluppare il livello applicativo. I Digital Twins rappresentano gli "abitanti" della replica del mondo reale su cui gli agenti svolgono le proprie azioni (esempio in ambito healthcare Figura 4.1).

Semplificando il concetto, Web of Digital Twins consente di replicare il mondo reale in un mondo digitale, rappresentando ogni asset appartenente alla realtà come Digital Twin nello spazio virtuale in modo tale da avere una copia sempre aggiornata su cui abilitare dei lavoratori che, osservando e prendendo decisioni, aiutano durante le attività svolte nel mondo reale. Ad esempio, un medico analizzando un paziente reale (quindi tutti gli asset fisici associati) può essere coadiuvato nelle proprie decisioni dagli agenti che "lavorando" nel mondo digitale analizzano la replica del paziente riuscendo a svolgere simulazioni, scenari what-if ecc.. che non sarebbero possibili per il medico.

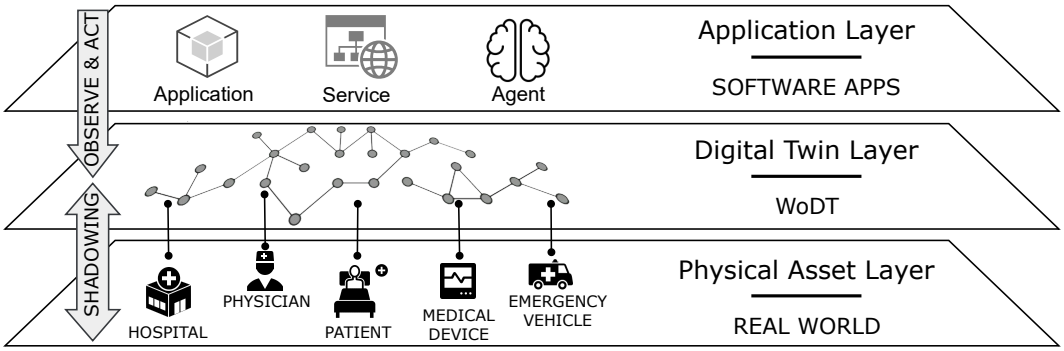

Figura 4.1: Visione a layer di Web of Digital Twins. Fonte: Web of Digital Twins [18]

Questo abilita il concetto di *extended mind* [19], in cui possiamo assistere ad un profondo impatto nei sistemi cognitivi umani e nella società. Grazie ai concetti abilitanti di Mirror world e Digital Twin, buona parte della nostra intelligenza potrebbe essere artificiale, o comunque aumentata artificialmente in real-time.

Come si può osservare, è qui di fondamentale importanza che ogni Digital Twin, non essendo più replicato in ogni organizzazione d'interesse, ma idealmente univoco, possa lavorare con diverse ontologie all'interno della stessa replica, quindi la piattaforma a supporto deve consentire la creazione di un "modello formato da più modelli". All'interno dell'healthcare, ad esempio, il Digital Twin di un'ambulanza può essere utile per diversi contesti applicativi: manutenzione da parte degli addetti e allocazione dei veicoli per la gestione di un'emergenza. In questi due casi saranno necessarie due viste sullo stesso Digital Twin completamente diverse, quindi due ontologie a seconda del caso d'uso. Al fine di raggiungere questi obiettivi, lo sviluppo dei modelli si basa su grafi di conoscenza (in modo simile, ma più complesso di ciò che viene offerto da Microsoft con Azure Digital Twins).

In una visione così aperta, con *asset fisico* non si intende solo ciò che ha effettivamente una rappresentazione concreta nella realtà. Include tutto ciò che è fisico come risorse o oggetti ma anche luoghi, persone e soprattutto attività e processi, similmente a quanto definito dal Digital Twin Consortium. In Web

of Digital Twins ogni asset strategico per un'organizzazione deve avere una sua controparte digitale (ad esempio Figura 4.2).

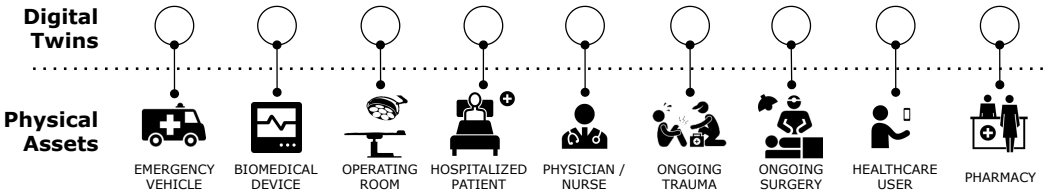

Figura 4.2: Esempi di asset all'interno di uno scenario healthcare. Fonte: Web of Digital Twins [18]

4.2 Modellazione in Web of Digital Twins

Per poter raggiungere questi requisiti, la rappresentazione di un Digital Twin può essere descritta attraverso una tupla $< I, P, M, S, C, T >$ che include le informazioni statiche e dinamiche del Digital Twin. La tupla contiene i seguenti elementi:

- Identità: è una caratteristica essenziale, già presente nelle prime versioni di Digital Twin.
- Binding: rappresenta il legame di un Digital Twin al proprio asset fisico (P).
- Modello: è una rappresentazione dell'asset fisico che include metadati rispetto allo stato S, al contesto C e al thread T. Inoltre, all'interno del modello viene incluso anche il comportamento atteso del Digital Twin per poter essere preso in considerazione all'interno delle simulazioni e previsioni.
- Stato: è l'effettivo modello del Digital Twin in termini di proprietà. Esse possono essere caratteristiche dirette dell'asset fisico, oppure dati derivati, aggregati o elaborati da computazioni.

- Contesto: rappresenta l'insieme delle relazioni che l'asset (quindi il suo Digital Twin) possiede con gli altri asset.

- Thread: utilizzato per rappresentare la sequenza di eventi che si susseguono durante il ciclo di vita del Digital Twin, includendo per ognuno un timestamp e una descrizione. È importante rappresentarli in maniera dettagliata in modo da poter ricostruire la storia del Digital Twin per l'esecuzione di analisi o simulazioni.

Grazie a questa modellazione, ogni cambio di stato o ogni evento dell'asset fisico rilevante per il modello viene rappresentato come un evento interno al Digital Twin che causerà un cambio di stato S o di contesto C e quindi una nuova situazione da salvare all'interno del thread T. Ciò suggerisce un approccio event-driven per il processo di *shadowing*.

Visto che gli agenti hanno bisogno di ragionare sulle informazioni dello stato e del contesto dei Digital Twins, è necessario un modello semantico che consenta di essere esplorato. Un *grafo di conoscenza*(knowledge graph) ha l'obiettivo di provvedere una descrizione semantica dello stato e delle relazioni dell'asset fisico (modellato come Digital Twin) all'interno del Web of Digital Twins. In questo modo possiamo creare un modello utilizzabile da applicazioni cross-domain che ragionano dinamicamente sullo stato corrente degli asset. Prendendo in considerazione sempre il Digital Twin dell'ambulanza, questo può essere utilizzato contemporaneamente dal sistema interno all'ospedale per pianificare l'attività di emergenza e dal sistema di Smart City al fine di regolare il traffico (semafori, ad esempio) in base alle esigenze, minimizzando il tempo di intervento.

Tutto il modello $< I, P, M, S, C, T >$, che come detto precedentemente può presentare diversi modelli e ontologie al suo interno, viene rappresentato attraverso un grafo di conoscenza. Esso può evolvere grazie al processo di shadowing e alle interazioni con gli agenti appartenenti al layer applicativo.

4.3 Considerazioni

Web of Digital Twins rappresenta una visione innovativa e contraddistinta da un'apertura che fin'ora non si trova in nessuna, al massimo della mia conoscenza, proposta a livello sia di ricerca che di mercato.

Comunque, si possono notare alcuni concetti minori già accennati o presi in considerazione, anche se in maniera meno approfondita, da altre soluzioni. Ad esempio, l'esposizione di interfacce diversificate a seconda del tipo di utilizzatore, discussa nella soluzione open-source in [6], viene ripresa qui con la possibilità di descrivere il modello attraverso più ontologie adattandosi al contesto d'uso. Inoltre, viene ripreso, anche se per ovvie necessità, la possibilità descritta in [17] di avere più sorgenti di input e più tecnologie in output per poter comunicare ed elaborare dati.

L'obiettivo finale di Web of Digital Twins potrebbe essere la creazione di un servizio alla pari del Web come lo conosciamo o addirittura del Web Semantico, in cui i protagonisti e le risorse sono proprio i Digital Twins. Infatti, Web of Digital Twins consente di definire Digital Twins con le loro proprietà, comportamenti, relazioni all'interno di un ecosistema distribuito su più organizzazioni ed interoperabile. Oltre a ciò, astrae il tutto introducendo un layer altamente interoperabile attraverso cui applicazioni, agenti e servizi possono lavorare con i Digital Twins come se fossero "abitanti" di quella che è la "copia di una porzione di realtà".

Il modello e i concetti descritti possono essere applicati e sviluppati in svariate modalità per progettare un'architettura a supporto. Infatti, nel prossimo capitolo descriverò quella che è la mia proposta.

Capitolo 5

Studio e ideazione di un middleware per Web of Digital Twins

Questa tesi ha l'obiettivo finale di progettare ed implementare un middleware all'interno di una proposta di architettura a supporto di Web of Digital Twins e verificarne l'applicabilità su un piccolo caso di studio.

Questo capitolo descrive due dei contributi che ho cercato di dare al concetto di Web of Digital Twins. Il primo contributo, descritto nella prima sezione, è l'analisi dei requisiti che un'architettura per Web of Digital Twins dovrebbe soddisfare e su cui basare gli eventuali sviluppi futuri. Il secondo, descritto nella seconda sezione, rappresenta la proposta architetturale vera e propria in modo da mettere in risalto problematiche e possibili soluzioni che ancora non sono state esplorate, proponendo primi passi per la risoluzione.

5.1 Requisiti del middleware

Come anticipato, il progetto di questa tesi verte sulla realizzazione di un middleware per Web of Digital Twins da inserire all'interno di un'architettura che consenta la creazione di un ecosistema di Digital Twins connessi ed inte-

roperabili. L'obiettivo è sopratutto quello di utilizzare le tecnologie e i Twin Builder esistenti rendendoli il più possibile interoperabili e quindi fare i primi passi verso la realizzazione di una soluzione orientata al concetto più puro di Web of Digital Twins.

Questo progetto non ha subito solo influenze da Web of Digital Twins; infatti, ho preso spunti da tutto ciò che ho studiato ed approfondito nel corso di questa tesi.
Ogni persona, ricercatore, studioso e organizzazione che propone soluzioni per questo concetto fornisce nuovi punti di vista, riflessioni e conclusioni che portano a comprendere sempre di più le potenzialità di questa tecnologia.

Grande influenza deriva perciò da National Digital Twins in quanto ha obiettivi simili a quelli del progetto, dalle proposte open source come ad esempio [17] e dalle visioni offerte dalle soluzioni attualmente presenti sul mercato come Microsoft Azure Digital Twins ed Eclipse Ditto.

La scelta di unire le conoscenze fornite da diverse fonti è, a mio parere, necessaria per iniziare a fare i primi passi verso la realizzazione concreta delle prime proposte di architettura. Infatti, sfruttando i prodotti che ad oggi sono a disposizione sul mercato e costruendo sopra di essi un middleware e un'infrastruttura che consenta di abbattere il siloing che contraddistingue quest'ultimi, è possibile comprendere più a fondo quali siano le problematiche e sopratutto espandere alcuni concetti cardine dei Digital Twins nel perseguire l'obiettivo di modellare la realtà che ci circonda seguendo l'idea di Mirror Worlds.

Questa strategia è simile a quella adottata da National Digital Twins con l'Information Management Framework, come si può vedere nella Figura 1.7. L'IMF ha il compito di unire i Twin Builder utilizzati all'interno delle varie organizzazioni, in quanto ad oggi, è impensabile costruire un unico Twin Builder con tutti i servizi annessi di Time Series Storage e Analsys ed obbligare tutte le organizzazioni ad utilizzarlo. Perciò è necessario fare un primo step per rendere compatibile e interoperabile ciò che ad oggi è utilizzato.

Parlando dei requisiti che il middleware e l'architettura a supporto dovranno soddisfare, si considera inizialmente, come detto nella sezione precedente, che un Digital Twin deve essere descritto attraverso una tupla $< I, P, M, S,$

C, T > e rappresentato all'interno di un *grafo di conoscenza*. Occorre perciò, comprendere come ogni componente della tupla deve essere implementato e sopratutto dove far risiedere e quale natura deve avere il grafo di conoscenza (centralizzato, federato, distribuito, ecc..).
Creando un'architettura composta da Twin Builder che utilizzano tecnologie diverse e che sono connessi tra loro attraverso vari middleware a formare un grande ecosistema di Digital Twins si va a creare una vera e propria rete, dal nome appunto *Web of Digital Twins*. Quindi, il singolo Digital Twin dovrà essere visto come un servizio, indirizzabile attraverso un proprio URI, a cui qualsiasi attore o agente (compatibilmente con i permessi) può accedervi per interagirvi e ottenerne informazioni. Come si può capire perciò, si vuole creare un sistema il più possibile interoperabile e standardizzato in cui il modo per accedere alle risorse (Digital Twin) è sempre lo stesso, indipendentemente dall'organizzazione a cui si accede o al Twin Builder utilizzato internamente da quella organizzazione. Dovranno essere superati gli specifici formati di richiesta e le specifiche API che contraddistinguono ogni Twin Builder costruendo sopra ad essi middleware in grado di offrire all'esterno un'unica interfaccia standard in tutto l'ecosistema. Per fare ciò i middleware dovranno adattare la richiesta compatibilmente alla tecnologia interna utilizzata.

L'ecosistema dovrà fornire i Digital Twins come un servizio, come se fossero delle risorse nel Web Semantico. Ogni organizzazione che compone l'ecosistema dovrà fornire la stessa interfaccia, rendendo di fatto impossibile per un agente o un attore comprendere la tecnologia utilizzata internamente. Questo rappresenta un notevole vantaggio per l'interoperabilità e per un utilizzo su larga scala in quanto consente ad ogni organizzazione di implementare il proprio concetto di Digital Twin, con le tecnologie che ritiene più opportune rispetto agli obiettivi e senza che lo specifico attore o agente debba conoscere le API specifiche per accedervi in quanto il tutto viene astratto da un'interfaccia comune offerta dal middleware.

Gli attori e gli agenti appena nominati faranno parte del layer applicativo che sarà possibile costruire sopra a questo ecosistema. Infatti, con la creazione di un ecosistema di Digital Twins a cui si accede in modo standardizzato si

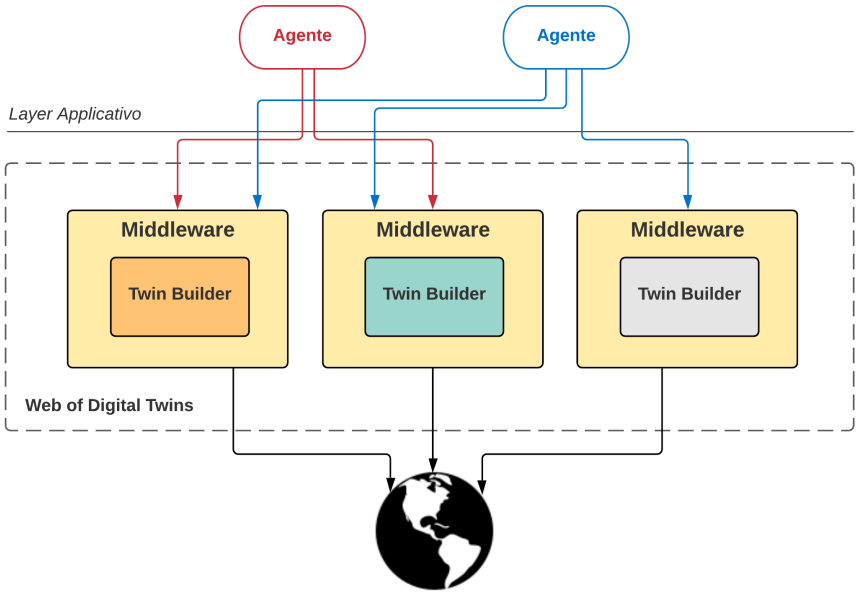

Figura 5.1: Astrazione del concetto abilitante di middleware nel contesto Web of Digital Twins

rende possibile lo sviluppo di agenti che, attraverso il Web of Digital Twins, riescono ad osservare il mondo reale abilitando una quantità di servizi fino ad oggi impensabili (Figura 5.1).

Al fine di abilitare gli agenti a lavorare con Web of Digital Twins è necessario individuare almeno un set di azioni minimo che compongono l'interfaccia verso il layer applicativo [18]:

- Creazione: deve essere possibile creare un'istanza di un modello (esistente) di Digital Twin che sia univocamente indirizzabile come se fosse una risorsa Web.

- Osservazione: fondamentale per effettuare il tracking continuo dell'evoluzione di un asset. Osservando tutti gli eventi prodotti da un Digital Twin si possono abilitare numerosi processi di analisi o predizioni completamente esterni all'ecosistema e gestiti individualmente dai vari agenti a seconda dei loro scopi.

- Query: necessarie per poter effettuare query sullo stato corrente del grafo di conoscenza e quindi dei rispettivi Digital Twins e asset associati. Le query possono essere a livello di singola organizzazione oppure inter-organizzazione. Nel primo caso l'agente sa a priori che i Digital Twins interessati sono tutti all'interno della stessa organizzazione e quindi può fare semplici query per ottenere le informazioni desiderate. Nel secondo caso occorre uno sforzo maggiore da parte dell'ecosistema e dell'architettura in quanto si vuole accedere ad informazioni sparse tra varie organizzazioni.

- Ottenimento risorsa: a volte è necessario ottenere tutte le informazioni di stato di un Digital Twin in modo da avere uno snapshot completo in un particolare istante temporale.

- Storia: i dati passati di un qualsiasi asset o di una qualsiasi risorsa sono di fondamentale importanza per le analisi e le predizioni.

- Predizione: grazie alla disponibilità dei dati storici e degli eventi in real-time è possibile fare predizione su diversi aspetti che caratterizzano ciascun Digital Twin. Qui, però occorre un ragionamento ad un livello di granularità maggiore per capire come assicurare interoperabilità (in quanto ogni classe di Digital Twin ha possibilità diverse in termini di predizione e qui, a differenza delle query, non abbiamo un linguaggio attraverso il quale esprimere la richiesta).

Per quanto riguarda l'ottenimento dello stato di un Digital Twin, prendendo spunto dallo stato dell'arte descritto nel capitolo precedente, è opportuno prevedere che un modello potrebbe essere descritto attraverso diverse ontologie al fine di adattarsi al caso d'uso dei client che desiderano accedervi.
Ad esempio, se avessimo un Digital Twin di un'ambulanza e due possibili agenti, da una parte l'ospedale e dall'altra il sistema di smart city, essi necessiteranno di tipologie di informazioni differenti e si potrebbe pensare di offrire interfacce diverse a seconda del caso d'uso. Questo è un concetto abilitante

molto importante che necessita ancora di notevoli analisi e studi per poter essere effettivamente implementato.

Un'altra caratteristica di base è lo *shadowing*. Lo shadowing è il processo continuo tramite il quale si mantiene sincronizzato un asset fisico con la sua replica digitale. Inizialmente il concetto prendeva in considerazione solamente gli aspetti statici del modello, quindi le sue proprietà. Per tutti gli altri aspetti, era l'agente che tramite richiesta alle API aggiornava la replica.

Questo è corretto in alcuni casi, mentre non perfettamente in linea con ciò che abbiamo descritto in altri, come ad esempio *le relazioni*. Infatti, nella maggior parte delle soluzioni attuali, le relazioni vengono create ed abilitate dagli attori esterni che, osservando il mondo, vengono a conoscenza di una qualche relazione e la "iniettano" nella replica. Mentre, se si ragiona sul concetto cardine e visionario di creare una replica digitale del mondo al fine di astrarre una componente software di elaborazione sopra di essa, la creazione o l'eliminazione di una relazione è come se fosse un aggiornamento di stato dell'asset, perciò deve *essere compreso nel processo di shadowing*.

In generale, tutte le operazioni di aggiornamento del modello statico o dinamico di un Digital Twin devono provenire dal processo di shadowing. Infatti, un agente esterno non può, a livello di stato dell'arte del concetto di Digital Twin, aggiornare i dati o le relazioni della replica digitale. Tutto ciò che un agente esterno è in grado di modificare sullo stato o sul comportamento deve rientrare in quella serie di funzionalità aggiuntive che prendono il nome di *Augmentation*.

Oltre a questo, ogni asset per essere considerato "valido" o "aggiornato" deve essere sincronizzato entro limiti temporali ben definiti. Un tipico esempio per comprendere l'importanza del fenomeno è quando ricerchiamo un negozio su un qualsiasi motore di ricerca per ottenere le informazioni sullo stato attuale, e una volta recatosi sul luogo vediamo che le informazioni erano errate (ad esempio troviamo il negozio chiuso per ferie). In questo caso, le informazioni non erano sincronizzate con la realtà, perciò non corrispondevano ad una replica valida dell'asset fisico.
Si può ben capire che nei contesti principali di applicazione dei Digital Twins

(healthcare, smart city, manifatturiero ...) questo è un episodio che non si deve verificare perché potrebbe essere seriamente dannoso per i processi coinvolti. Quindi è necessario che ogni Digital Twin venga descritto anche in termini di vincoli temporali e di qualità da rispettare, definendo metadati riguardanti la *fidelity*. Ad esempio, un Digital Twin per la tipologia di asset e di realtà che rappresenta potrebbe avere un periodo di aggiornamento massimo di tre secondi. In questo caso, se il Digital Twin non venisse aggiornato in tempo verrebbe impostato momentaneamente in uno stato di non validità. Ogni query che venisse fatta su quel Digital Twin, dovrebbe riportare un warning indicando che i dati forniti potrebbero non essere più in synch con l'asset fisico.
Non appena vengono forniti i nuovi dati il Digital Twin torna in uno stato di validità e quindi le informazioni di stato possono di nuovo essere considerate valide. Da notare che non per tutte le tipologie di Digital Twin è necessario un periodo di aggiornamento ristretto; infatti, per alcuni potrebbe essere sufficiente che i dati siano aggiornati entro minuti, ore o addirittura giorni.
Teoricamente la fidelity potrebbe essere assicurata a diversi livelli di granulosità. Infatti, si potrebbero inserire metadati riguardanti ogni elemento, proprietà o relazione e stabilire per ognuno di essi dei limiti temporali da soddisfare diversificati. Ad esempio nel Digital Twin di un'ambulanza i requisiti temporali dello stato del carburante e della sua posizione sono completamente differenti (per la posizione è necessario un aggiornamento molto più frequente per essere considerato adeguatamente informativo).

Oltre ai metadati, per assicurare il rispetto della fidelity è necessario un componente o un engine interno all'architettura che si occupi del controllo periodico del rispetto dei limiti temporali e dell'aggiornamento dello stato di validità di ogni Digital Twin. Questo componente deve essere in grado di accedere ai metadati riguardanti la fidelity e ai metadati riguardanti gli ultimi aggiornamenti in modo tale da poter verificare il soddisfacimento dei requisiti. L'engine potrebbe lavorare a due livelli: internamente ad ogni organizzazione, in modo che ogni nodo si controlli la validità di ogni Digital Twin individualmente, oppure come unico engine a livello di ecosistema che controlli tutti i Digital Twins presenti ed assicuri il rispetto dei limiti temporali. Nel secondo

caso è necessario che nel momento in cui si verifica un aggiornamento sulla validità di un Digital Twin questo venga segnalato al nodo dell'organizzazione corrispondente in modo che possa agire di conseguenza.

Grazie all'affidabilità dei dati memorizzati nei Digital Twins è possibile per gli agenti osservarne lo stato, richiederne particolari e sulla base di questi eseguire analisi che portano al delineamento di una serie di azioni da eseguire sul singolo Digital Twin. Un esempio tipico è la manutenzione o le azioni di correzione preventiva che un agente, osservando lo stato di un prodotto, può mettere in campo per salvaguardare la vita del prodotto stesso. Per fare ciò è necessario che ogni Digital Twin offra la proprietà descritta in [15] dell'*Augmentation* in modo da poter offrire una serie di funzionalità aggiuntive che un Digital Twin può soddisfare. Quest'ultime, non sono solamente servizi orientati alla modifica di dati per la creazione di dati derivati, ma possono inquadrare anche tutte quelle funzionalità atte e finalizzate all'esecuzione di specifiche azioni sugli asset associati ai Digital Twins. A supporto di tutto ciò è di fondamentale importanza che l'architettura che collega ogni organizzazione agli asset coinvolti nel processo di shadowing preveda collegamenti *bidirezionali* consentendo lo scambio continuo e simultaneo di dati in entrambe le direzioni.

Come si può notare, il processo di shadowing è uno dei più importanti servizi presenti in un ecosistema di Digital Twins. Infatti, consente ad un Digital Twin di essere chiamato tale ed abilita la maggior parte delle funzionalità aggiuntive. La sua complessità sta inoltre nel fatto che un Digital Twin o comunque un qualsiasi processo della realtà non viene aggiornato univocamente da un solo device o da un solo asset ma, il suo funzionamento, o nel caso dei Digital Twins, la sincronizzazione, deriva dall'utilizzo contemporaneo di più sorgenti di dati eterogenee, che quindi possono richiedere diverse tecnologie di collegamento o protocolli di comunicazione. Da questo deriva che il processo di shadowing deve, inoltre, occuparsi del raccoglimento dei dati in un unico formato. In particolare è di fondamentale importanza che nell'architettura ci sia un punto di raccolta dati che possa essere collegato a qualsiasi tipologia di asset, che possa utilizzare qualsiasi tecnologia, qualsiasi protocollo e qualsiasi formato dati, al fine di raccoglierli in un unico punto, validarli e spedirli come

un unico flusso di dati e di eventi al processo di shadowing interno all'organizzazione. Questo concetto è ispirato al concetto di adapter presente in [17] e visibile nella Figura 2.5 nel componente "Physical Interface".

A livello di modello, le soluzioni studiate fin'ora non consentono di formalizzare il comportamento di un Digital Twin. Tenendo in considerazione la natura condivisa e aperta di Web of Digital Twins sarebbe opportuno che gli agenti esterni potessero eseguire query o comunque essere a conoscenza delle possibilità compartimentali offerte da ogni Digital Twin. Per fare ciò è necessario specificare in modo formale il comportamento atteso e fornito da ogni asset e Digital Twin associato. Questo è sicuramente un requisito del middleware e dell'architettura nella sua versione finale anche se la sua realizzazione non rientrerà nel merito di questa tesi. Verrà fornita comunque una riflessione su una sua possibile implementazione.

Già da queste prime considerazioni si nota che alcuni requisiti sono simili a quelli del National Digital Twin, perciò può essere utile tenere in considerazione anche la necessità di avere dei componenti e delle formalità obbligatorie nello sviluppo dell'architettura:

- Catalogo: componente in grado di essere a conoscenza dell'intero ecosistema. Consente il "discovery" dei Digital Twins e consente di memorizzare informazioni di base su di essi.

- Upper-level ontology: rappresenta l'ontologia ad alto livello indipendente dal dominio applicativo. Sono tutti quei dati che un Digital Twin deve poter rappresentare solo per il fatto di essere tale. Quindi include almeno tutti i dati sulla fidelity, sulla qualità e sull'identità.

Inoltre seguendo [18] si aggiunge anche la necessità di:

- Definire i flow e le API: come detto, le interfacce e i flow interni a Web of Digital Twins devono essere standardizzati. Un agente deve poter seguire gli stessi flow e effettuare le stesse richieste (almeno quelle di base) a tutte le organizzazioni senza dipendere dalla tecnologia sottostante (e senza essere in grado di riconoscerla). È essenziale una descrizione

formale dei flow e soprattutto delle API che ogni interfaccia deve esporre per poter creare un ecosistema stabile nel tempo e su larga scala. La standardizzazione dei comportamenti accompagnata dall'interoperabilità in un sistema che punta a definire una rete di risorse è fondamentale (da notare che sono i punti di forza che hanno contraddistinto la riuscita del Web).

- Gestire le relazioni inter-organizzazione: essendo un ecosistema aperto è necessario consentire e quindi gestire in modo accurato le relazioni tra Digital Twins appartenenti ad organizzazioni diverse. Questo rappresenta un punto di svolta rispetto alle soluzioni presenti sul mercato in quanto abilita alla collezione e all'osservazione di fenomeni che fin'ora richiedevano l'impiego di tantissime risorse e il dispiegamento di infrastrutture ad-hoc.

Come già detto, in Web of Digital Twins, i Digital Twins vengono visti come servizi interni ad un ecosistema altamente interoperabile in cui gli agenti accedono ad ogni funzionalità tramite interfacce standard. Come ultimo ingrediente per la formalizzazione dell'ecosistema abbiamo la definizione dei linguaggi e dei formati con cui effettuare le richieste al fine di assicurare uno scambio di dati consistente in qualsiasi situazione. Nella visione proposta in questa tesi (che come detto riprende concetti da molte visioni esistenti) è di fondamentale importanza il grafo di conoscenza. I grafi di conoscenza sono un concetto abilitante anche nel Web Semantico, perciò Web of Digital Twins può trarre vantaggio da questa tecnologia ed utilizzare alcune tecniche già collaudate in questi contesti. Infatti, ogni Digital Twin rappresenta una risorsa individuabile tramite URI, quindi perfetta per essere utilizzata internamente a tecnologie per il Web Semantico. Si possono trarre vantaggi dal linguaggio di codifica e scambio dati RDF [5], da quelli per definire gli schemi delle risorse RDFS [3] e OWL [1] oltre che dal linguaggio di query SPARQL [2]. RDF verrà utilizzato ogni qual volta occorre restituire i dati riguardanti lo stato di un Digital Twin. Mentre, per quanto riguarda le query, queste dovranno essere specificate dagli agenti esterni in SPARQL e le varie organizzazioni dovran-

no tornare il risultato in uno dei formati compatibili con la recommendation [2] (JSON, XML, CSV, TSV); io mi soffermerò principalmente su SPARQL-JSON [20]. Ovviamente non tutte le tecnologie utilizzate dalle organizzazioni potranno dialogare direttamente con questi linguaggi, ad esempio Microsoft non ha alcun tipo di supporto né per RDF né per SPARQL. Perciò, saranno necessari, a livello di middleware, dei convertitori ad-hoc che consentano di poter utilizzare verso l'esterno i linguaggi standard del Web Semantico (assicurando interoperabilità ed indipendenza dalla tecnologia). Ad esempio, Microsoft Azure Digital Twins per poter effettuare le query sul proprio grafo di conoscenza richiede un linguaggio SQL sviluppato ad-hoc. Quindi, si rende necessario un convertitore che consenta di passare da una query SPARQL ad una query Azure Digital Twins SQL. Occorre essere in possesso di strumenti, tecnologie e algoritmi in grado di interpretare e comprendere a fondo una query SPARQL ed elaborare i risultati restituendoli in un formato compatibile con la recommendation, come ad esempio SPARQL-JSON. Inoltre, anche per quanto riguarda i dati sullo stato è necessario creare dei convertitori che, a partire dai risultati forniti dalle API delle tecnologie utilizzate internamente, restituiscano i dati in formato RDF. Anche per quanto riguarda l'amministrazione e la creazione di nuovi modelli da parte di agenti esterni (ancora in fase di analisi e ricerca) è molto plausibile l'utilizzo di tecnologie provenienti dal Web Semantico come appunto RDFS ed OWL. In aggiunta, potrebbe essere utile definire uno schema per la validazione dei dati in SHACL [14]. Questi schemi dovranno essere convertiti nei linguaggi di modellazione utilizzati dai Twin Builder[1]. Nelle richieste per cui non è possibile utilizzare linguaggi appartenenti al Web Semantico occorre definire dei formati di richiesta descritti in un linguaggio standard. A questo fine, possono essere utili JSON ed XML in modo da assicurare l'interoperabilità e l'indipendenza dai linguaggi di programmazione.

Per soddisfare i requisiti dell'ecosistema, con tutte le conversioni a carico del middleware, si potrebbe pensare di prendere ispirazione dall'architettura

[1]Esempio di un tool di Microsoft per convertire RDFS/OWL in DTDL: `https://github.com/azure-samples/rdftodtdlconverter/tree/main/`

proposta in [18] in cui sono presenti degli adapter sia a livello di comunicazioni esterne (interfacce con gli agenti), sia a livello di processo di shadowing (interfacce con il sistema di raccolta dati e gli asset). Questi adapter nel nostro caso avranno dei sotto-componenti dedicati alla conversione dei dati nei formati appropriati. Inoltre, è opportuno prevedere una visione a componente in cui si rende possibile l'installazione di adapter aggiuntivi, come se fossero "estensioni", senza la necessità di cambiare il codice degli altri componenti come accade in [17] per gli adapter nella parte di comunicazione con gli asset.

Queste interfacce, che consentono di astrarre dal Twin Builder interno, dovrebbero essere costruite in modo tale che siano indipendenti dalla particolare istanza di quella tecnologia e che quindi il loro deploy sia semplice e veloce in ogni organizzazione che utilizza la stessa architettura.

È comunque di fondamentale importanza che qualsiasi attore, asset, sistema di data ingestion o in generale agente si interfacci in qualsiasi modo con un'organizzazione non sia in grado di riconoscere la tipologia di tecnologia o le tecniche utilizzate internamente e non sia dipendente da alcuna parte di essa. L'obiettivo è costruire diversi livelli di astrazione, mediati tramite software e capaci di astrarre dalle tecnologie al fine di mantenere vivo l'aspetto semantico e concettuale dei Digital Twins ed utilizzarli come se fossero delle vere e proprie porzioni di realtà.

Il middleware e ogni componente dell'architettura dovrebbe essere sviluppato sfruttando le "best-practise" ad oggi presenti in termini di sviluppo orientato ai *microservizi* e il più possibile *event-driven*. Questi due approcci consentono di ottenere diversi vantaggi. Il primo, l'approccio a microservizi, consente di poter sviluppare il tutto come un insieme di componenti ognuno dei quali con una funzionalità specifica attraverso l'utilizzo di diversi linguaggi di programmazione e soprattutto permettendo, una volta applicato in un contesto reale, di gestire la scalabilità per ogni componente in modo indipendente, assicurando una notevole capacità di adattamento al traffico. Inoltre, la gestione dell'infrastruttura event-driven consente ai microservizi di comunicare tra di loro mantenendo il disaccoppiamento offerto da questa tecnologia.

Infine, sarà sicuramente necessario un componente di Logging e Analysis

dell'infrastruttura che consenta di analizzarne l'utilizzo, gestire le situazioni di errore e condurre root cause analysis.

Altri punti di sviluppo e scenari non implementati saranno riflessioni per lavori futuri sul progetto.

5.2 Proposta architetturale

Dopo aver descritto i requisiti del middleware e dell'architettura a supporto per Web of Digital Twins, in questa sezione descriverò la mia proposta mostrando tutto ciò che ho progettato partendo dall'architettura fino alle specifiche di tutte le API e degli eventi coinvolti nei processi senza entrare nei dettagli implementativi. Infine, verranno descritti i principali flow che caratterizzano i principali casi d'uso del sistema.

Come per i requisti, l'architettura subisce notevoli influenze da tutto ciò che è stato studiato e approfondito durante il corso di questa tesi, in particolare da National Digital Twin, dalle architetture offerte dal mondo della ricerca e ovviamente da Web of Digital Twins.

L'obiettivo da perseguire è la realizzazione di un ecosistema di Digital Twins derivato dalla creazione di una rete di organizzazioni, ognuna delle quali può utilizzare le tecnologie e i Twin Builder che ritiene più consoni, mediati da middleware opportuni che rendano l'accesso e l'utilizzo dell'ecosistema standardizzato. Si vuole cercare di portare un primo e semplice contributo alla progettazione e all'implementazione calata in un contesto di disponibilità tecnologiche reale del concetto di Web of Digital Twins [18].

Il pattern architetturale proposto segue l'approccio consigliato da National Digital Twin per la realizzazione dell'Integration Architecture [13]. Infatti, è previsto un core, chiamato *Web of Digital Twins Core* e un insieme di nodi chiamati *Web of Digital Twins Node*. Ogni nodo si collega al core e, almeno per ora, non sono previsti collegamenti tra nodi. I *Nodi*, in questo contesto, rappresentano le varie organizzazioni che desiderano partecipare attivamente all'ecosistema. Essi, per partecipare, dovranno essere circondati, come descritto precedentemente, da interfacce che assicurino l'osservanza degli standard di

interoperabilità decisi a livello di ecosistema. In questo modo sarà possibile fornire a tutti i client e a tutti i device la stessa interfaccia di utilizzo abilitando la possibilità di vedere *ogni* Digital Twin come se fosse un servizio a cui si accede nello stesso modo con cui si accede ad una qualsiasi altra risorsa sul Web, qualsiasi sia la sua organizzazione, qualsiasi tecnologia nasconda sotto di sé. Il *Core* fornirà tutto il supporto e la gestione necessaria per poter orchestrare un ecosistema che, per ora, non è distribuito nel vero senso del termine. Si cerca di creare un ecosistema che appaia distribuito per compensare le mancanze tecnologiche, al massimo della mia conoscenza, che sono presenti oggi (in quanto calando in un contesto reale si vogliono utilizzare i Twin builder ad oggi presenti sul mercato). Esso perciò si dovrà occupare, come il "Core IMF" in National Digital Twin, del tracciamento di tutti i Digital Twins presenti nell'ecosistema, verificare la loro validità, dare una prima possibilità di relazioni inter-organizzazione e quindi promuovere il buon funzionamento dell'ecosistema.

Una descrizione dell'architettura ad alto livello è offerto nella Figura 5.2. Si può notare come le tuple $< I, P, M, S, C, T >$ siano memorizzate all'interno dei nodi, perciò ogni organizzazione gestisce e memorizza i Digital Twins con le tecnologie che ritiene più opportune. Infatti, si vede come ogni nodo abbia un macro-componente *Twin Builder* in cui risiede tutta la varietà tecnologica e l'apporto che ogni organizzazione prevede al fine di implementare la propria soluzione basata su Digital Twins. Dopodiché, come già descritto, la tecnologia interna viene avvolta da adapter ed interfacce in modo da assicurare l'interoperabilità all'interno dell'ecosistema. Queste macro-interfacce sono il *Physical Asset Adapter* a livello di shadowing e l'*External World Adapter* a livello di comunicazione con gli agenti esterni.

L'architettura cerca di affacciarsi agli approcci a microservizi ed event-driven consigliati oggigiorno per la progettazione e lo sviluppo di architetture simili. In questa visione astratta, il nodo appare come un componente semplice composto dai seguenti macro-componenti:

- Twin Builder: è la parte che l'organizzazione può progettare completa-

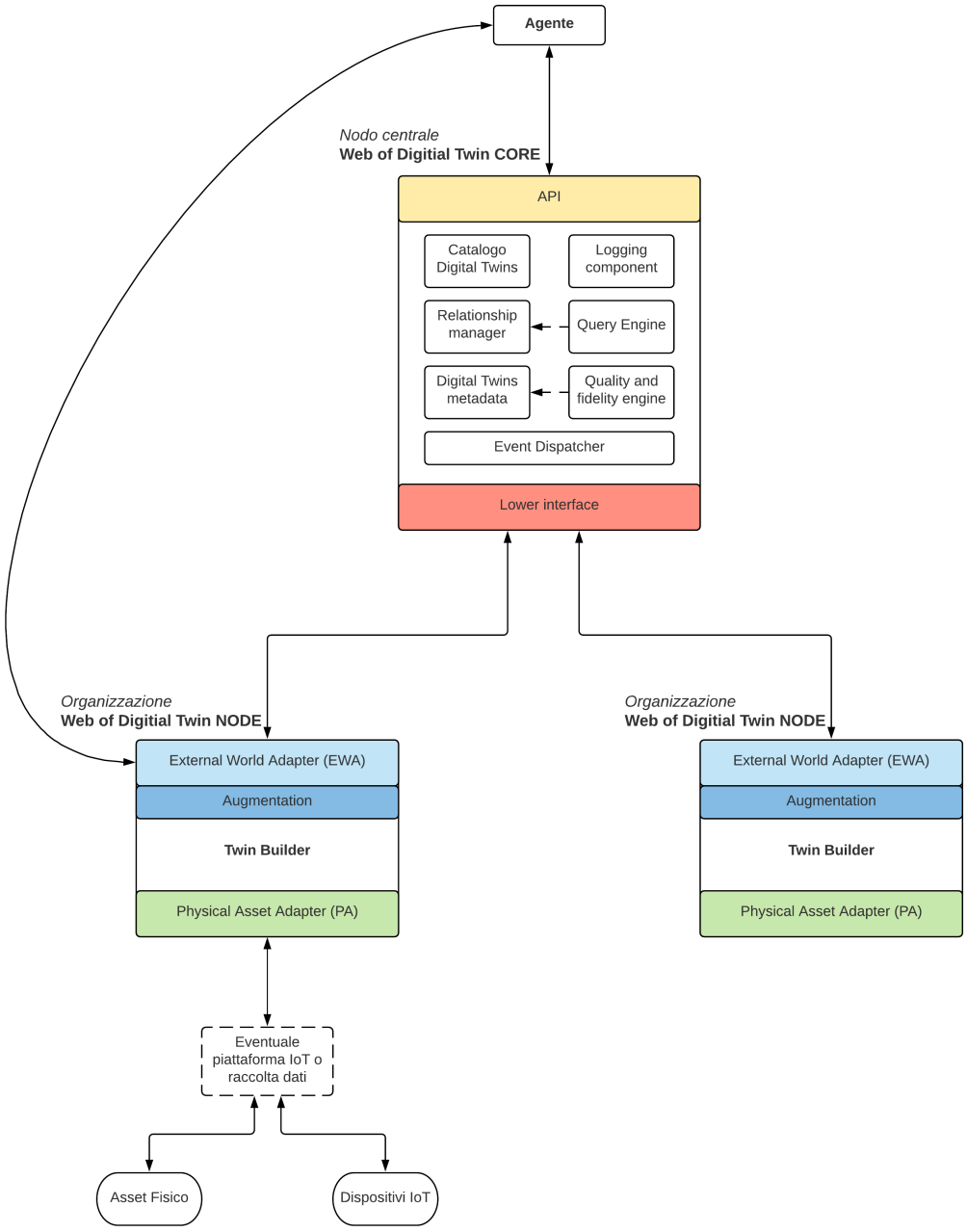

Figura 5.2: Visione astratta dell'architettura proposta

mente a piacere con l'obiettivo di soddisfare i propri scopi rientrando allo stesso tempo nei requisiti minimi applicativi richiesti dall'ecosiste-

ma. Essa può essere realizzata con qualsiasi tecnologia e architettura con deploy locale, sul cloud o ovunque si desideri. Questo è un punto molto importante di questa visione, cioè consentire a tutte le organizzazioni di mantenere o comunque progettare la propria struttura indipendentemente dal fatto che dovranno partecipare all'ecosistema senza necessità di adeguarsi o utilizzare tecnologie specifiche imposte. Ad esempio, si può pensare di utilizzare la soluzione di Microsoft, oppure Eclipse Ditto assieme a tutti gli strumenti aggiuntivi come InfluxDB o Elasticsearch. Per tutta la parte di adattamento e "creazione dell'interoperabilità a livello di ecosistema" ci penseranno le interfacce e gli adapter in cui questo macro-componente è racchiuso.

- Physical Asset Adapter (PA): è l'interfaccia che consente di offrire il servizio di conversione degli eventi, contenenti i dati raw provenienti dal processo di shadowing, nei dati aderenti al modello del Digital Twin opportuno e memorizzato all'interno del Twin Builder. Quindi, esso si dovrà occupare di aggiornare le istanze di Digital Twins e di completare il processo di shadowing sia dal punto di vista della modellazione statica (proprietà) sia di quella dinamica (relazioni, come detto precedentemente fanno parte, nella visione adottata, del processo di shadowing). Per fare ciò è necessario, oltre ad adattare i dati al modello di Digital Twin, anche utilizzare le API o l'SDK offerti dalle tecnologie utilizzate nel macro-componente "Twin Builder" individuato.

- External World Adapter (EWA): rappresenta il punto di contatto con gli agenti esterni. Fornisce tutte le API necessarie per interagire con il nodo ed ottenere le informazioni desiderate sui Digital Twins. Le API che espone sono frutto della progettazione delle richieste a livello di ecosistema. Quindi, ogni organizzazione, attraverso l'External World Adapter, dovrà esporre le stesse API in modo tale che un agente possa utilizzare qualsiasi organizzazione (nodo) allo stesso modo, indipendentemente dalle tecnologie utilizzate internamente. Perciò, per soddisfare le richieste dovrà adattarle e convertirle per ottenere i dati, come il Physical Asset

Adapter, attraverso le API e l'SDK offerti dal Twin Builder. Infine, esso si dovrà occupare di mantenere aggiornato il Web of Digital Twins Core in merito agli eventi interni all'organizzazione attraverso il canale bidirezionale che li collega (descritto successivamente).

- Augmentation: è necessario un componente o comunque una parte del sistema che si occupi di gestire l'augmentation. Questa è una questione affrontata da pochissimi Twin Builder o soluzioni pubbliche, in quanto necessita di ancora tanto studio anche dal punto di vista della modellazione. Però, a livello concettuale, vedo questa funzionalità come un componente definito che deve assicurare certe funzionalità.
  Sull'augmentation verrà successivamente fatta una piccola riflessione.

I dati provenienti dagli asset fisici provengono da un componente di raccolta dati il quale deve essere in grado, come da requisito, di utilizzare qualsiasi tecnologia o protocollo necessario per comunicare con gli asset e deve collezionare i dati in modo tale da fornire al nodo un unico bus di eventi descritti in modo uniforme e indipendente dalla sorgente. Questo significa che il Physical Asset Adapter non deve avere la necessità di adattare il proprio funzionamento a seconda della tecnologia o del protocollo adottato dalla sorgente, bensì non deve neanche essere in grado di capirlo, deve vedere gli eventi di shadowing come un unico flusso in cui i dati sono definiti seguendo degli standard in modo da poterli processare velocemente. Per questo motivo anche il componente di raccolta dati può essere sviluppato a piacere dall'organizzazione. L'unico requisito è che esponga i dati nella maniera prevista dall'ecosistema al Physical Asset Adapter e che abbia un collegamento bidirezionale con quest'ultimo. Idealmente potrebbe essere realizzato dall'organizzazione (come in questa tesi, al fine di mostrare le potenzialità) oppure potrebbe essere adottata una soluzione già esistente, anche in cloud (come Microsoft Azure IoT Hub), che consenta l'ottenimento delle stesse finalità e un meccanismo per esportare i dati nel formato desiderato verso un endpoint (in questo caso appunto il Physical Asset Adapter).
Questa architettura lascia volutamente ampia scelta implementativa anche da

questo lato, definendo solamente le regole essenziali per poter partecipare all'ecosistema.
Mentre, per quanto riguarda il core esso è composto dai seguenti componenti:

- Lower Interface: è l'interfaccia di comunicazione con i Web of Digital Twins Node. Raccoglie gli eventi provenienti dalle varie organizzazioni e li fa gestire all'Event Dispatcher. Inoltre, essendo il canale di comunicazione con i nodi bidirezionale, viene sfruttata sempre questa interfaccia per tutti gli eventi elaborati dal core e da spedire alle singole organizzazioni. Gli eventi necessari per la gestione dell'ecosistema verranno illustrati successivamente.

- Event Dispatcher: riceve gli eventi provenienti dalle organizzazioni attraverso la Lower Interface e a seconda dell'evento richiama il componente destinato ad elaborarlo.

- Catalogo Digital Twins: esso si occuperà, come nel caso dell'Integration Architecture in National Digital Twin, di tracciare tutti i Digital Twins presenti a livello di ecosistema. Essi verranno rappresentati con un modello standard che contiene i metadati associati al Digital Twin attraverso quella che viene definita una sorta di "upper-level ontology". Il catalogo dovrà essere informato quando viene creato o eliminato un Digital Twin all'interno delle varie organizzazioni. Questi sono solo due degli eventi di cui si parlava nei componenti precedenti.

- Digital Twins metadata: per poter assicurare la fidelity e la qualità dei dati associati ai Digital Twins è necessario prevedere dati aggiuntivi con cui ogni Digital Twin deve essere descritto. Oltre a ciò è necessario memorizzare lo stato di un Digital Twin in termini di validità, cioè se i dati di una replica possono essere considerati validi e quindi sincronizzati, oppure non validi e quindi fuori sincronizzazione (in quanto hanno superato i tempi massimi di aggiornamento). Ho deciso di creare logicamente un componente a parte al fine di sottolineare la sua importanza, però

in un'implementazione reale potrebbe essere tranquillamente un sotto componente del Catalogo.

- Quality and fidelity engine: è l'engine che consente di ottenere effettivamente il controllo sulla fidelity e sulla qualità dei dati associati ai Digital Twins. Esso controlla continuamente il rispetto dei limiti temporali di aggiornamento delle repliche digitali e in caso ne aggiorna lo stato di validità. Non appena un Digital Twin passa da uno stato di validità ad un altro viene generato un evento che poi, passando per la Lower Interface, viene spedito all'organizzazione in cui quel Digital Twin effettivamente risiede. Questo permette all'organizzazione di specificare, nel momento in cui viene fatta una richiesta sullo stato di un Digital Twin, anche lo stato di validità di quei dati, in modo che il client sia consapevole e soprattutto sappia se quei dati sono frutto di una sincronizzazione considerata ancora valida oppure no.

- Relationship manager: gestisce le relazioni inter-organizzazione presenti all'interno dell'ecosistema. Quindi permette di creare una relazione tra due Digital Twins che risiedono in due organizzazioni diverse consentendo un primo passo verso la creazione di un ecosistema unico. È un componente tutt'ora in fase di analisi e studio in quanto utilizzare le tecnologie odierne e allo stesso tempo consentire relazioni che vanno fuori dal "contenitore" che ogni servizio costruisce internamente ad ogni organizzazione è una sfida tutt'ora aperta (ad esempio i Digital Twins all'interno di Microsoft Azure Digital Twins possono essere messi in relazione solamente con altri Digital Twins interni alla stessa istanza di servizio).

- Query Engine: consente l'esecuzione di query per ottenere i metadati sui Digital Twins dell'ecosistema e per scoprire le relazioni inter-organizzazione.

- API: fornisce il punto di accesso per le operazioni a livello di ecosistema. Quindi fornisce accesso al servizio di discovery di un Digital Twin, al

servizio di query e al servizio per ottenere i metadati di un Digital Twin (nel caso fosse necessario).

- Logging component: tenendo in considerazione che i nodi e il core comunicano ad eventi, un componente di logging che effettui log su tutto ciò che succede all'interno dell'ecosistema può rivelarsi di fondamentale importanza per diversi scenari. Primo su tutti utilizzare i dati per eseguire analisi sull'utilizzo dell'ecosistema e quindi apportare migliorie nei punti più deboli. Inoltre, è uno strumento di fondamentale importanza nel caso di guasti per effettuare una root cause analysis.

Si nota da questa prima descrizione ad alto livello che sono presenti due componenti che gestiscono le relazioni tra Digital Twins: il Twin Builder interno ad ogni organizzazione e il Relationship manager nel Core. Il motivo di questa scelta è l'impossibilità di memorizzare tutte le relazioni dei propri Digital Twins internamente all'organizzazione dovuto alle restrizioni imposte da alcuni Twin Builder presenti ad oggi sul mercato. Se analizziamo la soluzione dominante per ora, cioè quella di Microsoft, essa consente di creare relazioni solamente con Digital Twins interni a quella particolare istanza del servizio, perciò va a creare una sorta di contenitore (che appunto è il grafo di conoscenza che si crea in Azure Digital Twins) dove vengono memorizzati tutti i dati e le relazioni dei Digital Twins per il contesto d'uso. Il problema è che per specificare una relazione dobbiamo indicare l'id di un Digital Twin interno all'istanza stessa, perciò non potremmo indicare un'URI di un Digital Twin che si trova in un organizzazione diversa. Né tanto meno potremmo utilizzare una proprietà per "emulare" una relazione in quanto in questo modo non potremmo più avere relazioni miste (cioè sia con Digital Twins interni, sia con Digital Twins esterni all'organizzazione).
Una soluzione diametralmente opposta sarebbe quella di gestire tutte le relazioni attraverso un unico grafo di conoscenza all'interno del Core. Questa soluzione eliminerebbe qualsiasi problema tecnologico in quanto le relazioni verrebbero gestite in unico luogo, però solleva altre problematiche. Innanzitutto, è necessario che le organizzazioni segnalino la necessità di creare una

relazione al Core tramite il canale di comunicazione presente tra l'EWA e la Lower Interface e che poi il Core notifichi l'organizzazione riguardante l'esito. Quindi è necessario un meccanismo per assicurare consistenza nelle richieste. Inoltre, è necessario che il Core sia a conoscenza dei modelli di ogni Digital Twin in modo che possa stabilire se una relazione può essere creata oppure no. Questo problema potrebbe essere risolto lasciando l'onere del controllo alle organizzazioni e supporre che tutto ciò che arriva al Core sia già consistente con la modellazione data dalle organizzazioni. Questa soluzione è tutt'ora in fase di studio in quanto è quella che sembra dare i risultati migliori a discapito di alcune problematiche tutt'ora da risolvere.
Perciò, quello che ho fatto in questa architettura è unire le due soluzioni in modo tale da fare un primo passo verso la risoluzione e sopratutto analizzare i vantaggi e gli svantaggi di entrambi gli approcci. In questa soluzione ogni organizzazione gestisce solamente le relazioni che coinvolgono Digital Twins interni ad essa, mentre il Core gestisce tutte le relazioni che coinvolgono Digital Twins appartenenti a due organizzazioni diverse. Quindi è necessario che durante il processo di shadowing le organizzazioni riconoscano se la relazione coinvolge Digital Twins interni o esterni e opportunamente segnalino il Core, tramite evento, nel caso in cui sia necessario definire una relazione inter-organizzazione.

La scelta di lasciare ampio spazio e libertà nelle tecnologie da utilizzare internamente ai servizi e definire solamente gli standard e l'architettura generale con i componenti dei middleware è voluta al fine di creare un approccio che non sia solamente concettuale, ma un qualcosa che possa essere effettivamente realizzato considerando organizzazioni che hanno già soluzioni IoT o soluzioni Digital Twins interne a regime. Quindi non si vuole creare uno svantaggio nel partecipare all'ecosistema, ma si vuole creare un valore aggiunto per la società. Il principio per cui i protagonisti non sono obbligati ad utilizzare la stessa tecnologia, ma anzi la caratteristica di essere aperti a supportare le tecnologie esistenti sviluppando opportuni middleware è un approccio che, per ora, ritengo di fondamentale importanza.

Al fine di descrivere i Digital Twins dell'intero ecosistema all'interno del

Catalogo presente nel Core ho creato una modellazione molto semplice che rimanda l'idea di "upper-level ontology". L'obiettivo è quello di creare una rappresentazione di un Digital Twin slegata dal dominio applicativo al fine di rappresentare tutti i dati comuni che caratterizzano una replica digitale solo per il fatto di esserla. È composta dalle seguenti proprietà:

- Id del Digital Twin: è l'identificatore utilizzato internamente all'organizzazione in cui è memorizzato e gestito.

- URI dell'organizzazione: è l'uri di base dell'organizzazione che gestisce il Digital Twin in questione. Corrisponde al prefisso che viene utilizzato per accedere alle APIs. Concatenando l'URI di base e l'id del Digital Twin si deve ottenere l'URI di accesso allo stato del Digital Twin stesso. Inoltre questa concatenazione rappresenta un identificatore univoco a livello di ecosistema.

- Classificazione: descrive i casi d'uso e i settori di interesse del Digital Twin in questione. È una classificazione *multi-tag* tramite il quale ad un singolo Digital Twin possono essere associati più settori di interesse. Ad esempio, un'Ambulanza potrebbe essere inserita sia nel settore "healthcare" sia in quello "city". In questo modo, un'ipotetica query potrebbe richiedere quali sono gli attori importanti sia per la città, sia per l'healthcare oppure solo per uno dei due ed otterrebbe in tutti i casi le Ambulanze. La classificazione viene eseguita a livello di modello e non di istanza, perciò nel caso d'esempio tutte le Ambulanze saranno importanti in quei settori. Un ulteriore sviluppo o punto di considerazione potrebbe essere sollevato nel spostare la classificazione a livello di singola istanza.

- Stato: rappresenta lo stato di validità del Digital Twin. Quindi indica se esso è in-sync oppure off-sync rispetto alla porzione di realtà che rappresenta.

- Ultimo update: metadato sulla data e l'ora di ultimo aggiornamento.

- Requisiti di aggiornamento: rappresenta l'intervallo temporale entro cui il Digital Twin deve essere aggiornato per essere considerato in-sync. È un *primo passo* verso il concetto puro di *fidelity*. Questo può essere diverso istanza per istanza e quindi non dipende dal modello come nel caso della classificazione.

- Data e ora creazione: data e ora di creazione del Digital Twin.

- Tipo di Digital Twin: è la tipologia di Digital Twin. È profondamente diverso dal concetto di classificazione descritto prima. Riguarda la tipologia di Digital Twin così come intesa nel National Digital Twin programme: Descrittivo, Informativo, Predittivo, Prescrittivo, Cognitivo. *Descrittivo* si occupa di collezionare e "visualizzare dati", *Informativo* dà la possibilità di generare degli insights aggregando e analizzando i dati, *Predittivo* consente inoltre di eseguire predizioni di fenomeni, *Prescrittivo* aggiunge la possibilità di elaborare delle proposte attraverso l'analisi dei dati ed infine *Cognitivo* il quale attraverso i sistemi a supporto riesce a prendere autonomamente delle decisioni e condurre delle azioni.
  Oltre a questa, si potrebbero aggiungere delle informazioni anche riguardo al tipo di asset rappresentato: elemento fisico, processo/attività o realtà "virtuale" (cioè qualcosa che esiste nella realtà solamente attraverso sistemi che aumentano la nostra percezione di realtà).

Una volta fissati i requisiti e l'architettura generale occorre espandere ulteriormente la proposta ad un livello di dettaglio tale per cui l'implementazione può seguire direttamente lo schema ma, allo stesso tempo, è indipendente da eventuali linguaggi di programmazione, framework o librerie.
Per quanto riguarda le tecnologie, ho deciso di progettare un Web of Digital Twins Node che basasse la sua infrastruttura interna, quindi il suo macrocomponente Twin Builder, sulla soluzione offerta da Microsoft la quale è stata oggetto di particolare interesse durante questa tesi.
Partendo proprio dal Nodo realizzato, si può notare come il numero di sotto

componenti sia cresciuto in modo sostanziale. Nella Figura 5.3 si può osservare la progettazione in dettaglio del nodo.

Il macro-componente Twin Builder è formato da diversi servizi in cloud di Microsoft i quali, disposti in questo modo, permettono di modellare Digital Twins, effettuare richieste, query, osservare in real-time l'evoluzione di un Digital Twin, memorizzare la storia dei Digital Twins ed infine eseguire analisi interne all'organizzazione. Per quanto riguarda la predizione ancora non è stata progettata un'interfaccia standard a livello di ecosistema, però, comunque sia, questa architettura consente di avere componenti di predizione da utilizzare internamente all'organizzazione. Sistemi di predizioni possono essere creati anche esternamente al nodo andando ad includere le analisi e gli algoritmi di apprendimento all'interno di un agente, il quale raccoglie i dati attraverso il processo di osservazione in real-time di un Digital Twin fornito dal nodo stesso.

Un aspetto importante da tenere in considerazione è l'insieme di funzionalità e delle tecniche di modellazione necessarie all'interno dell'ecosistema. Si deve pensare ad un sotto-insieme di funzionalità minime che devono essere coperte dalle tecnologie interne. In questa prima versione, nella modellazione di un Digital Twin vengono considerate solamente le *proprietà* e le *relazioni*, le quali ci consentono di modellare gran parte delle situazioni. Questo significa che il Twin Builder di ogni nodo dovrà almeno fornire questi strumenti di modellazione.
Inoltre, nel caso in cui la tecnologia utilizzata supporti ulteriori modellazioni rispetto a quelle minime, queste, all'esterno dell'ecosistema, non potranno essere utilizzate (invece internamente possono essere tranquillamente sfruttate, intendendo proprio dai software che lavorano internamente all'organizzazione e quindi non coloro che utilizzano le API dell'ecosistema).
Questa osservazione si riversa su Azure Digital Twins, dove il DTDL fornisce, oltre alle proprietà e alle relazioni, anche altre possibilità di modellazione, come i *componenti* e i *dati telemetrici*. Quest'ultimi non potranno essere utilizzati a livello di ecosistema.

Come anticipato, ho deciso di progettare un Web of Digital Twins Node che

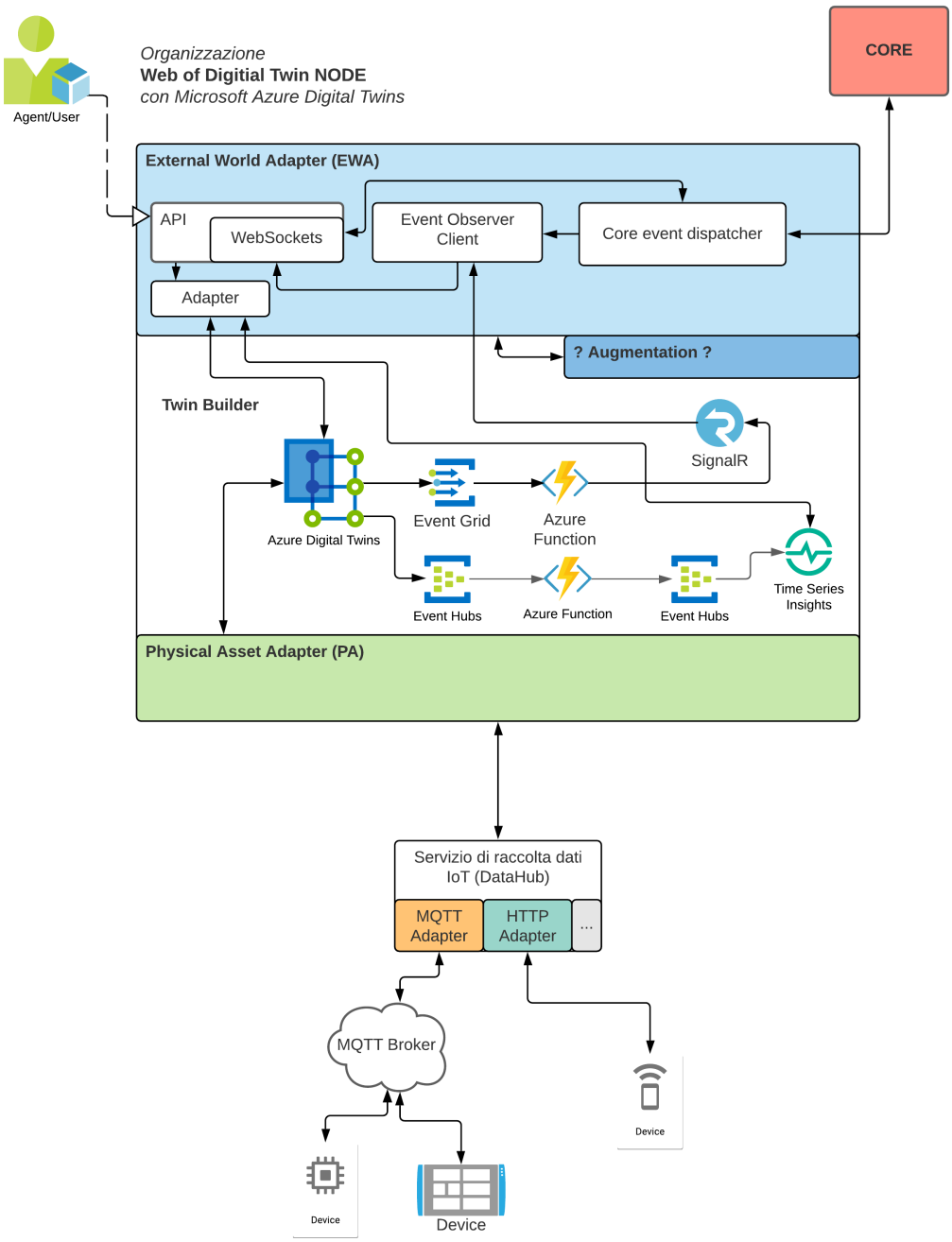

Figura 5.3: Visione di dettaglio della proposta di un nodo basato su Microsoft Azure Digital Twins

sfrutti la soluzione proposta da Microsoft: *Azure Digital Twins*. Come detto durante la descrizione delle soluzioni presenti sul mercato, Azure Digital Twins autonomamente non supporta tutte le funzionalità richieste in questo ambito, perciò, è necessario utilizzare anche altri servizi, sempre in casa Microsoft.

Prima di parlare dei servizi aggiuntivi, è necessario descrivere come è possibile collegare Azure Digital Twins a servizi esterni per poter giustificare l'architettura creata. Gli eventi di Azure Digital Twins producono *notifiche*, le quali consentono ai servizi collegati di venire a conoscenza di ciò che sta accadendo all'interno della nostra istanza del servizio. Le notifiche, per poter giungere ai servizi collegati, vengono passate alle *event route*, le quali forniscono i dati agli *endpoint* supportati da Azure Digital Twins e a cui poi possono essere connessi i vari servizi che necessitano di ricevere quelle notifiche. Gli eventi che Azure Digital Twins rende disponibile verso l'esterno sono:

- Aggiornamento del Digital Twin: riguardano le notifiche sull'aggiornamento di ogni proprietà dei Digital Twins. Viene generata una notifica contenente tutti i dettagli dell'aggiornamento.
- Ciclo di vita del Digital Twin: riguardano le notifiche sulla creazione e l'eliminazione dei Digital Twins all'interno del servizio.
- Relazioni: riguardano le notifiche sulla creazione, eliminazione e modifica delle relazioni tra Digital Twins all'interno del servizio.
- Telemetria: riguardano i dati telemetrici, non presi in considerazione dall'ecosistema.

Le event routes consentono di spedire questi eventi, prodotti dai Digital Twins interni ad Azure Digital Twins, ad altri componenti dell'architettura o a servizi esterni attraverso gli endpoint. Azure Digital Twins supporta tre tipologie di endpoint: Event Hub, Event Grid e Service Bus. Ognuno di questi servizi agisce come intermediario nella comunicazione degli eventi tra Azure Digital Twins e il resto dei componenti (o servizi) che necessitano di questi dati. Quindi, sostanzialmente, Azure Digital Twins rende disponibile l'evento alle event

routes, dopodiché le event routes che soddisfano il filtro (in quanto ogni route può specificare un filtro sugli eventi che desidera ricevere) spediscono l'evento all'endpoint associato il quale mette a disposizione i dati ai servizi ad esso collegati. È per questo motivo che il servizio Azure Digital Twins, all'interno dell'architettura, è collegato ai servizi Event Grid ed Event Hubs all'interno della pipeline per fornire i servizi di tracking e threading.

Per quanto riguarda la pipeline per ottenere il servizio di tracking essa è formata da tre servizi principali: Event Grid, Azure Function e SignalR. L'Event Grid, come anticipato, è uno degli endpoint supportati da Azure Digital Twins. Esso è un servizio che consente di realizzare applicazioni con architettura basata sugli eventi. Viene creato un topic in cui inviare i dati (quindi in cui Azure Digital Twins, in questo caso, invia gli eventi), dopodiché si possono specificare dei trigger o delle sottoscrizioni che ascoltano la ricezione dei dati e li elaborano.
L'Azure function invece, è una soluzione che consente di specificare porzioni di codice ed eseguirle serveless con la possibilità di scaling automatico. Nessun deploy, nessuna manutenzione, solo codice su cloud che può eseguire liberamente. Nel capitolo successivo verrà descritto il dettaglio implementativo, per ora mi limito a dire che le Azure functions vengono solitamente "svegliate" per l'esecuzione da opportuni "Trigger" e possono specificare dei "Bindings" per ottenere i dati in input oppure per descrivere risorse a cui spedire i dati in output. Sostanzialmente il trigger è l'elemento che causa l'esecuzione della funzione attraverso un evento (in cui è possibile trasportare un payload), il binding in input rappresenta un ulteriore sorgente di dati in input eventualmente necessaria, mentre il binding in output rappresenta il servizio a cui restituire i dati in output. Essendo una porzione di codice scritta nei più comuni linguaggi di programmazione, C#, Java, Javascript, Python ecc.., può richiamare tutte le SDK e le API di qualsiasi servizio, rendendo apparentemente inutile il binding in output. In realtà è una funzionalità molto interessante e molto comoda nel momento in cui si lavora con altri servizi Microsoft.
Infatti, sono disponibili numerosi Trigger e Binding, tra cui:

- Trigger Event Grid: è possibile quindi eseguire la funzione al ricevimento di un evento sull'Event Grid.

- Binding in output SignalR: è possibile spedire i dati a SignalR semplicemente dalla Azure Function.

Infine, SignalR è un servizio che consente la creazione di canali di comunicazione bidirezionali in real-time sfruttando HTTP. Consente quindi di spedire contenuti ai client collegati in real-time senza che i client necessitino di eseguire polling al server o eseguire richieste HTTP particolari.

Tenendo in considerazione che l'interfaccia verso l'esterno (External World Adapter, EWA) deve essere a conoscenza di tutti gli eventi interni al proprio Twin Builder per poi fornirli agli agenti che intendono osservare uno specifico Digital Twin, è necessario seguire i seguenti passaggi:

1. creare una event route che non abbia alcun tipo di filtro sugli eventi di Azure Digital Twins e quindi sia in grado di raccoglierli tutti;

2. creare un Event Grid Topic da associare alla route appena creata in modo da poter spedire gli eventi fuori dal servizio Azure Digital Twins;

3. creare una Azure Function basata su Event Grid Trigger in modo tale che ogni ad nuovo evento di Azure Digital Twins la funzione venga eseguita;

4. all'interno della Azure Function elaborare il payload dell'evento aggiungendo tutte le informazioni necessarie ed eseguire il binding della funzione al servizio SignalR. In questo modo, non appena si scatena un evento, l'Azure Function esegue, elabora i dati e li spedisce a SignalR;

5. configurare un componente del middleware, in particolare l'Observer Event Client, come client per l'istanza SignalR in modo tale da poter ricevere tutti gli eventi della tecnologia sottostante. Da questo momento in poi il middleware, attraverso i suoi componenti, sarà in grado di elaborare i dati prodotti dai Digital Twins e di metterli a disposizione dei client interessati.

Tramite queste tecniche è possibile creare un layer di astrazione tra l'agente che fa richiesta di tracking al nodo e la tecnologia interna. Il middleware, in particolare l'External World Adapter con i suoi componenti (descritti in seguito), media le comunicazioni tra l'agente e il Twin Builder utilizzato internamente per poter fornire i dati nel formato standard richiesto dall'ecosistema.

La pipeline per ottenere il servizio di threading, o più semplicemente per memorizzare la storia di un Digital Twin, è formata da un Event Hub che lavora come endpoint di Azure Digital Twins e una Azure Function che prende i dati in ingresso tramite un Event Hub Trigger e spedisce i dati in uscita attraverso un binding ad un altro Event Hub il quale opera da input per il servizio Time Series Insights.
Event Hub è un servizio di big data streaming che può processare milioni di eventi al secondo. In questo caso, come endpoint di Azure Digital Twins, poteva essere utilizzato nuovamente Event Grid, però ho inserito questo servizio per mostrare come, in tale circostanza, siano intercambiabili. Invece, lato Time Series Insights deve essere utilizzato obbligatoriamente in quanto quest'ultimo supporta input solamente da Event Hub o IoT Hub. Time Series Insight è un servizio che consente di memorizzare, processare, visualizzare ed eseguire query su dati espressi in serie temporali. È fornito di un potente sistema di API che consente di ottenere i dati e di fare query specificando l'istante o il periodo temporale per cui ricevere i dati, perciò è adatto a memorizzare e fornire le funzionalità utili per la gestione della storia di un Digital Twin.

I dettagli implementativi e i vari linguaggi utilizzati per lo sviluppo verranno forniti nel capitolo successivo.

Al fine di utilizzare i servizi di Microsoft è necessario, come anticipato, convertire i dati nei formati opportuni, da un lato per poter fare richiesta alle tecnologie interne, dall'altro per poter restituire i dati nei formati e nei linguaggi opportuni all'agente che si interfaccia all'ecosistema. Di questo se ne occuperanno i vari adapter e converter all'interno del macro-componente External World Adapter (EWA).

L'External World Adapter è formato da diversi componenti ognuno dei

quali riveste un ruolo fondamentale sia per il nodo sia per la comunicazione e lo scambio di dati con il core, perciò è importante generalmente a livello di ecosistema. Esso è formato da:

- API: sono le API tramite il quale i vari agenti effettuano richieste all'organizzazione. Questo componente racchiude dentro di se anche un sotto componente che gestisce le Web Socket. Le Web Socket vengono utilizzate per consentire ad un agente di osservare un particolare Digital Twin. Esse vengono create convertendo direttamente la connessione aperta per la richiesta in modo tale da poter aggiornare l'agente in real-time.

- Adapter: considerando che il Twin Builder non utilizza né lo stesso formato né gli stessi linguaggi richiesti dall'ecosistema è necessario un componente che consenta di adattare i contenuti e le richieste. In particolare, pensando al Twin Builder Microsoft, è necessario tradurre le query SPARQL che provengono dagli agenti nel formato richiesto da Microsoft Azure Digital Twins il quale è un SQL ad-hoc. Dopodiché, anche i risultati saranno da convertire in quanto Azure Digital Twins restituisce i risultati in un JSON con formato ad-hoc (e che quindi non segue nessuno standard), mentre a livello di ecosistema, per assicurare tutte le proprietà definite nei requisiti, è necessario restituire i risultati in un formato compatibile con la recommendation SPARQL. In particolare, per questa proposta di architettura ho deciso di utilizzare SPARQL-JSON [20] come formato del risultato da ritornare agli agenti. Perciò, è necessario un ulteriore convertitore che esegua questa operazione di conversione. Inoltre, anche nel momento in cui si ottengono i dati di un Digital Twin, il servizio di Microsoft ritornerà un JSON che sarà da convertire in RDF al fine di fornire i dati sullo stato di un Digital Twin compatibilmente con i requisiti definiti.

- Event Observer Client: è necessario un componente che si colleghi al meccanismo che fornisce gli eventi del Twin Builder in real-time e che li spedisca al componente API per far si che vengano inviati nelle web

sockets interessate. In questo modo, non si creeranno tanti client per il servizio interno di osservazione quante sono le web sockets interessate, ma si creerà un unico client, interno all'EWA, il quale, comunicando tramite il bus ad eventi interno all'architettura, spedirà i dati al componente API che poi replicherà i dati su tutte le web sockets interessate.
Nel caso del nodo Microsoft, l'Event Observer Client si collegherà al servizio SignalR definito precedentemente.

- Core event dispatcher: gestisce il canale bidirezionale di comunicazione con il Core. È il componente che agisce da intermediario per la realizzazione di molti servizi all'interno dell'ecosistema. Infatti, esso gestisce l'invio e la ricezione di tutti gli eventi rilevanti per le funzionalità interne ed esterne. Il Core per poter assicurare il corretto funzionamento dei suoi componenti, come il Catalogo, il Quality and Fidelity engine e il Relationship Manager, necessita di essere a conoscenza di ogni creazione, eliminazione, modifica dei Digital Twins e anche della creazione o eliminazione di qualsiasi relazione tra Digital Twins. Inoltre, il Nodo per poter informare gli agenti sullo stato di validità di un Digital Twin necessita di ricevere i dati sullo stato di validità dal Core. Dunque, questo componente attraverso uno scambio continuo di eventi mantiene sincronizzate ed aggiornate le varie componenti dell'architettura.

Come si vede dallo schema questi componenti sono collegati tra di loro attraverso un bus che lavora ad eventi. I collegamenti sono necessari per assicurare il raggiungimento dei vari obiettivi descritti per i componenti. Ad esempio, il collegamento tra il componente API e il Core event dispatcher è necessario per aggiornare da una parte il Core sulla creazione, eliminazione dei Digital Twins e dall'altra per aggiornare il componente API sulla validità dei Digital Twins; oppure il collegamento tra l'Event Observer Client e il componente API è necessario per ottenere i dati raccolti dal primo e renderli disponibili per le Web Sockets.
Questi collegamenti e l'architettura event-driven assicurano il disaccoppiamen-

to dei servizi e la possibilità, in un deploy reale, di scalare ognuno di essi a seconda delle necessità e dei carichi di lavoro.

Gli eventi scambiati tra il Nodo e il Core necessitano di una formalizzazione in modo tale che tutte le organizzazioni comunichino con il Core allo stesso modo. È importante che sia formalizzato non solo il linguaggio, ma anche il formato degli eventi e i contenuti così che il Core possa interpretarli in un unico modo. Gli eventi che il Nodo può inviare al Core sono i seguenti:

- Creazione di un Digital Twin: quando un agente richiede la creazione di un Digital Twin all'interno di un'organizzazione, il nodo deve notificare il Core in modo tale che possa aggiornare il Catalogo ed in modo da consentire il controllo da parte del Quality and fidelity engine.

```
{
   "org" : "http://localhost:8080/api/twins/",
   "type" : "CREATE",
   "id" : "Ambulance1",
   "fidelity" : "1000",
   "classes" : ["healthcare", "city"]
}
```

Listato 5.1: Payload evento di creazione di un Digital Twin

In questa prima versione, a scopo dimostrativo, viene incluso anche un campo che specifica l'URI dell'organizzazione che invia il messaggio. In particolare, corrisponde al prefisso dell'URL tramite cui, concatenando l'id del Digital Twin, è possibile contattare l'API per l'ottenimento dello stato (che corrisponde, come detto nei requisiti, all'identificatore a livello di ecosistema). In contesti reali, si potrebbe pensare che all'avvio della socket di comunicazione tra il nodo e il core si effettui un processo iniziale di sincronizzazione e una sorta di handshake in cui vengano scambiati i dati di configurazione e si effettui la sincronizzazione.
Dopodiché, all'interno del messaggio viene specificato il tipo di evento, in questo caso "CREATE", l'id del Digital Twin creato, la fidelity deside-

rata dall'agente e le classi di appartenenza dell'istanza. Il campo fidelity è opzionale e se non viene specificato allora il Digital Twin verrà sempre considerato valido.

- Eliminazione di un Digital Twin: quando un agente richiede l'eliminazione di un Digital Twin all'interno di un'organizzazione, il nodo deve notificare il Core in modo tale che possa aggiornare il Catalogo e tutti i servizi associati.

```
{
   "org" : "http://localhost:8080/api/twins/",
   "type" : "DELETE",
   "id" : "Patient1"
}
```

Listato 5.2: Payload evento di eliminazione di un Digital Twin

Viene specificata l'organizzazione, sempre per il motivo precedente, il tipo dell'evento inviato ("DELETE") e l'id del Digital Twin da eliminare.

- Aggiornamento di un Digital Twin: quando, grazie al processo di shadowing, viene aggiornato un Digital Twin il nodo lo deve notificare al Core in modo tale che possa aggiornare i dati riguardanti l'ultima modifica della replica e possa monitorare lo stato di validità attraverso il Quality and Fidelity engine (infatti quest'ultimo confronta la data e ora di ultimo aggiornamento e i requisiti temporali dati dalla fidelity).

```
{
   "org" : "http://localhost:8080/api/twins/",
   "type" : "UPDATE",
   "id" : "Mission1"
}
```

Listato 5.3: Payload evento di aggiornamento di un Digital Twin

Abbiamo gli stessi campi dell'eliminazione. Ovviamente il tipo sarà "UPDATE".

- Creazione di una relazione inter-organizzazione: quando, grazie al processo di shadowing, viene creata una nuova relazione tra Digital Twins appartenenti ad organizzazioni diverse, in particolare uno appartenente all'organizzazione d'interesse e l'altro esterno, è necessario richiederne la creazione, per i motivi discussi prima, al Core.

```
{
   "org" : "http://localhost:8080/api/twins/",
   "type" : "CREATEREL",
   "source" : "Mission1",
   "relname" : "hasPatient",
    "target" : "http://hostname/api/twins/Patient123"
}
```

Listato 5.4: Payload evento di creazione di una relazione inter-organizzazione

I campi sono simili a quelli dei messaggi precedenti. È presente l'organizzazione, il tipo di evento ("CREATEREL"), l'id della sorgente della relazione (che appunto deve essere il Twin interno all'organizzazione), il nome della relazione che lega i due Digital Twins e l'URI che identifica a livello di ecosistema il Digital Twin target.

- Eliminazione di una relazione inter-organizzazione: quando, grazie al processo di shadowing, viene eliminata una relazione tra Digital Twins appartenenti ad organizzazioni diverse, come precedentemente uno interno all'organizzazione di interesse e uno esterno, è necessario richiederne l'eliminazione al Core.

```
{
   "org" : "http://localhost:8080/api/twins/",
   "type" : "DELETEREL",
   "source" : "Mission1",
```

```
   "relname" : "hasPatient",
    "target" : "http://hostname/api/twins/Patient123"
}
```

Listato 5.5: Payload evento di eliminazione di una relazione inter-organizzazione

I campi sono gli stessi di quelli necessari per la creazione, solamente che il tipo specificato è "DELETEREL".

Mentre, per quanto riguarda gli eventi che il Core invia al Nodo, per ora, di particolare importanza c'è solamente un evento, quello per cui viene notificato il cambiamento dello stato di validità di un Digital Twin monitorato dal Quality and Fidelity Engine. Questo evento ha il seguente formato:

```
{
   "type" : "VALIDITY",
   "id" : "Patient1",
   "valid" : true
}
```

Listato 5.6: Payload evento di cambiamento di stato di validità di un Digital Twin

I campi di questo evento, sono simili a quelli precedenti. Abbiamo il tipo di evento impostato a "VALIDITY", l'id del Digital Twin interessato (il Core riesce a specificare direttamente l'id del Digital Twin interno all'organizzazione) e lo stato di validità del Digital Twin attraverso un valore booleano: quando vale *true* allora è valido (quindi è sincronizzato con la realtà entro i limiti temporali stabiliti) mentre quando è *false* è in uno stato di non validità (quindi non è sincronizzato con la realtà entro i limiti temporali stabiliti).

Questi appena definiti sono gli eventi principali che coinvolgono Nodi e Core. Sicuramente, proseguendo con le analisi e con gli studi si troverebbero nuove interazioni interessanti e utili. Tuttavia, l'aspetto realmente importante nella definizione della proposta architetturale non è trovare tutti gli eventi pos-

sibili o ogni minima sotto-funzionalità necessaria, ma stabilire la base per poter poi espandere il concetto in modo sostenibile, come in questo caso è importante stabilire la *necessità* di una forma di standardizzazione nella descrizione degli eventi e le informazioni minime necessarie.

Oltre agli eventi, è molto importante stabilire le API che il componente API dell'External World Adapter dovrà mettere a disposizione degli agenti. Qui è di fondamentale importanza che tutti i Nodi rispettino il formato, i parametri e i codici di risposta in modo tale che gli agenti possano effettuare richieste e ricevere risposte con le stesse modalità a tutti i Nodi, indipendentemente dalla tecnologia utilizzata. Le API sono le seguenti:

- Ottenimento stato completo di un Digital Twin: un agente potrebbe voler richiedere lo stato completo di un Digital Twin in termini di proprietà. L'URL della richiesta deve avere il seguente formato:

  GET http://*dominio*/api/twins/{id}

  L'id del Digital Twin viene specificato come path parameter. L'URL così formato corrisponde all'URI del Digital Twin utilizzato per identificarlo a livello di ecosistema.
  Il risultato di questa richiesta viene restituito in RDF in modo tale da rispettare i requisiti dell'architettura e rappresentarlo in un formato comprensibile a tutti in modo standardizzato. Inoltre, gli *status code* che la richiesta deve restituire sono i seguenti:

  - 200 (OK): risorsa presente e restituita, il body contiene la descrizione del Digital Twin (quindi della risorsa richiesta) in RDF.
  - 404 (Not Found): risorsa non trovata, id del Digital Twin non presente all'interno dell'organizzazione.
  - 500 (Internal Server Error): si può verificare quando il software lato server che esegue la richiesta del Twin Builder fallisce per qualche problema interno.

- Osservazione di un Digital Twin: l'agente ha la possibilità di osservare un Digital Twin al fine di ottenere tutti gli eventi che lo coinvolgono all'interno dell'ecosistema: creazione, eliminazione, relazioni, aggiornamenti ecc.. L'URL della richiesta deve avere il seguente formato:

  GET http://*dominio*/api/twins/{id}/events

  L'id del Digital Twin viene specificato come path parameter. Non appena la richiesta viene inviata al Nodo, se il Digital Twin è presente nell'organizzazione, la richiesta HTTP viene convertita in una Web Socket al fine di poter scambiare i dati (ottenuti, nel caso del nodo in questione, dall'Event Observer Client il quale a sua volta ottiene i dati dal servizio SignalR) in real-time in modo continuativo e senza effettuare polling. Gli eventi vengono inviati in formato JSON. In particolare abbiamo cinque eventi principali:

  - Creazione del Digital Twin: viene data la possibilità di osservare un Digital Twin prima della sua effettiva creazione. Questo perché si possono verificare situazioni in cui si è interessati ad essere notificati nel momento in cui un particolare Digital Twin viene creato. Il payload dell'evento contiene l'id del Digital Twin interessato e il tipo dell'evento impostato a "*DigitalTwin.created*".
  - Eliminazione del Digital Twin: per lo stesso motivo precedente, viene data la possibilità di osservare anche il momento in cui un Digital Twin viene eliminato. Il payload dell'evento contiene nuovamente l'id del Digital Twin interessato e il tipo dell'evento impostato a "*DigitalTwin.deleted*".
  - Aggiornamento del Digital Twin: di fondamentale importanza è l'evento di aggiornamento dello stato di un Digital Twin. Nel payload viene specificato l'id, il tipo "*DigitalTwin.updated*" e viene specificato l'aggiornamento di stato attraverso un oggetto in JSON Patch[2]

[2] `http://jsonpatch.com/`

che consente di specificare cambiamenti di stato in formato JSON. Esso permette inoltre di evitare di dover inviare tutto lo stato del Digital Twin ogni volta e specificare solo ciò che effettivamente è stato aggiornato.

- Creazione di una relazione che coinvolge il Digital Twin: ogni qualvolta viene creata una relazione che coinvolge il Digital Twin viene inviato un evento in cui è specificato l'id del Digital Twin sorgente e target (il Digital Twin osservato potrebbe essere sia l'uno che l'altro), il nome della relazione e ovviamente il tipo di evento, impostato a "*Relationship.created*".
- Eliminazione di una relazione che coinvolge il Digital Twin: ogni qualvolta viene eliminata una relazione che coinvolge il Digital Twin viene inviato un evento in cui sono specificati esattamente gli stessi dati della creazione di relazione, tranne il tipo che viene impostato a "*Relationship.deleted*".

Se la richiesta va a buon fine non viene ritornato alcun *status code* ma viene semplicemente creata la Web Socket. Invece, se la richiesta fallisce viene ritornato lo *status code* 400 (Bad request) in quanto significa che c'è qualche problema nella richiesta o 500 (Internal Server Error) nel caso ci siano stati problemi lato server.

- Creazione di un Digital Twin: richiesta di creazione di un'istanza di Digital Twin appartenente ad un tipo (modello) esistente.

POST http://*dominio*/api/twins/{type}

Il tipo del Digital Twin da creare viene specificato come path parameter. Invece, il body della richiesta contiene:

- Id: campo "dtId", è l'id da assegnare al Digital Twin creato.
- Intervallo di aggiornamento: campo "fidelity", è l'intervallo temporale massimo che può passare tra due aggiornamenti. Serve a

definire la fidelity e quindi il limite temporale entro cui il Digital Twin può essere considerato valido oppure no. Viene specificato in millisecondi.

Inoltre, gli *status code* che la richiesta deve restituire sono i seguenti:

- 201 (Created): risorsa creata (Digital Twin), viene impostato inoltre l'header della risposta *Location* all'URI del Digital Twin appena creato.
- 400 (Bad Request): richiesta mal formattata, manca l'id del Digital Twin oppure la richiesta non è in JSON valido.
- 409 (Conflict): id già presente all'interno dell'organizzazione. Quando l'id è già presente impedisce la creazione ed imposta l'header *Location* all'URI del Digital Twin esistente.
- 500 (Internal Server Error): quando la richiesta fallisce a causa di errori lato server.

- Cancellazione di un Digital Twin: consente di eliminare un'istanza di Digital Twin.

DELETE http://*dominio*/api/twins/{id}

L'id del Digital Twin viene specificato come path parameter e il body della richiesta è vuoto. Inoltre, gli *status code* che la richiesta deve restituire sono i seguenti:

- 204 (No content): risorsa eliminata, nel body non ritorno lo stato della risorsa prima di essere eliminata, quindi è per questo che utilizzo 204 al posto di 200.
- 404 (Not Found): risorsa inesistente, id del Digital Twin non presente all'interno dell'organizzazione.
- 500 (Internal Server Error): quanto la richiesta fallisce a causa di errori lato server.

- Query locale: l'agente ha la possibilità di eseguire delle query sul grafo di conoscenza interno all'organizzazione al fine di ottenere informazioni riguardanti lo stato e le relazioni.

  POST http://*dominio*/api/twins/sparql

  La query, come da requisito, viene specificata in SPARQL. In accordo con la recommendation W3C [2] utilizzo il metodo POST per la richiesta di esecuzione di una query.
  Ho scelto POST invece di GET in quanto con la richiesta GET è necessario specificare la query come query string, risultando, a mio parare, di più difficile utilizzo.
  Il body della richiesta contiene la query senza codifiche aggiuntive e senza parametri aggiuntivi. Il risultato, secondo la recommendation e i requisiti, viene espresso in SPARQL-JSON[20]. Inoltre, gli *status code* che la richiesta deve restituire sono i seguenti:

  - 200 (OK): la query è stata processata correttamente e viene restituito il risultato.
  - 400 (Bad Request): la query ha degli errori e non può essere elaborata.
  - 500 (Internal Server Error): quando la richiesta fallisce a causa di errori lato server.

I dati con cui aggiornare i Digital Twins provengono dal processo di shadowing. In questo processo vengono coinvolti diversi componenti dell'architettura.

Iniziando dal basso, i dati vengono prodotti dai vari asset fisici, sensori, device e dai vari sistemi a supporto delle controparti fisiche. Questi devono poter comunicare i dati al sistema in modo indipendente dalla particolare tecnologia, quindi senza essere obbligati ad utilizzare protocolli o tecnologie particolari.

Per questo motivo è presente un servizio di raccolta dati che viene chiamato *Data Hub*. Il Data Hub colleziona i dati provenienti da tutti gli asset

fisici che sono coinvolti nel processo di shadow di qualche Digital Twin interno all'organizzazione. Esso dovrà consentire, come anticipato, di potersi collegare attraverso qualsiasi tecnologia e protocollo sia necessario.
Per far fronte a questo requisito è costruito in modo tale che sia componibile da vari adapters i quali comunichino con il Data Hub ad eventi in modo standardizzato. Gli adapters non sono altro che piccoli moduli che si preoccupano solamente di permettere il collegamento con gli asset attraverso la propria tecnologia e inviare i dati raccolti al Data Hub.
Per verificare le potenzialità di questo approccio, in cui potremmo idealmente avere decine e decine di adapter ognuno per una tecnologia diversa, ho creato due adapter: un adapter MQTT e un adapter HTTP.
Per poter lavorare con un adapter MQTT è necessario disporre di un broker, che agisca da intermediario nella comunicazione, ed inoltre di una definizione formale dei topic da utilizzare. Tenendo in considerazione che i device possono essere centinaia di migliaia ho deciso di non creare un topic per device, ma di lavorare per tipo di Digital Twin per cui effettuare lo shadow. Ogni device pubblicherà i propri dati nel topic:

wodt/{orgX}/tipo Digital Twin

dove {orgX} indica l'identificativo dell'organizzazione e {tipo Digital Twin} indica il tipo di Digital Twin per cui si stanno inviando dati. Questo consente di evitare la proliferazione di topic e soprattutto lato adapter di poter raccogliere tutti i dati in unico punto attraverso la subscribe con l'uso di *wildcard*. Infatti l'adapter potrà semplicemente effettuare la subscribe a:

wodt/{orgX}/+

in modo da ottenere i dati di ogni tipo possibile.

Invece, per quanto riguarda l'adapter HTTP esso consentirà ai device di poter specificare i dati attraverso delle semplici richieste ad API. Ho creato due API principali per ora, in modo da evidenziare anche qui la possibilità di espansione.

- Update di un Digital Twin: aggiorna le proprietà dello stato di un Digital Twin.

  PUT http://*dominioshadow*/api/twins/{tipo}/{id}

  Viene specificato sia il tipo che l'id del Digital Twin in modo da rendere consapevoli i layer soprastanti (principalmente il Physical Asset Adapter) del modello da adottare nella conversione dei dati nel modello di riferimento.
  Il body della richiesta contiene le proprietà del Digital Twin da aggiornare associate ai corrispettivi valori. Inoltre, gli *status code* che la richiesta deve restituire sono i seguenti:

  - 200 (OK): dati ricevuti e processati.
  - 400 (Bad Request): errore nella richiesta.
  - 500 (Internal Server Error): quando la richiesta fallisce a causa di errori lato server.

- Creazione di una relazione: comunica la creazione di una relazione tra due Digital Twins. Il Digital Twin sorgente deve appartenere all'organizzazione d'interesse, mentre il Digital Twin target può essere sia interno che esterno.

  PUT http://*dominioshadow*/api/twins/{id sorgente}/relationships/create

  L'id sorgente viene specificato attraverso il path parameter. Invece, il body della richiesta contiene il nome della relazione e l'identificativo del Digital Twin target. Nel caso sia un Digital Twin interno all'organizzazione verrà indicato semplicemente l'id. Mentre, nel caso sia un Digital Twin esterno all'organizzazione verrà indicato l'URI che lo identifica a livello di ecosistema. Inoltre, gli *status code* che la richiesta deve restituire sono i seguenti:

  - 200 (OK): relazione creata correttamente.
  - 400 (Bad Request): errore nella richiesta.
  - 500 (Internal Server Error): quando la richiesta fallisce a causa di errori lato server.

- Eliminazione di una relazione: comunica l'eliminazione di una relazione tra due Digital Twins. Il Digital Twin sorgente deve appartenere all'organizzazione d'interesse, mentre il Digital Twin target può essere sia interno che esterno.

PUT http://*dominioshadow*/api/twins/{id sorgente}/relationships/delete

Utilizzo anche qui "PUT" e non "DELETE" in quanto la relazione non viene considerata una risorsa, ma una componente dinamica dello stato, perciò sia la creazione che l'eliminazione le considero come aggiornamenti del Digital Twin stesso. Come nella richiesta di creazione, viene specificato l'id sorgente attraverso il path parameter e nel body vengono inseriti gli stessi dati. Anche gli *status code* restituiti sono i medesimi.

I dati scambiati sono espressi tutti in JSON.

Quindi, il Data Hub avvia e gestisce tutti gli adapter collegati. In questo modo, gli adapter si limitano a ricevere i dati che vengono subito spediti al Data Hub. È sempre nel Data Hub che avvengono i controlli di validità dei dati (campi obbligatori e conformità) in modo tale da centralizzare i controlli ed evitarne la ripetizione in ogni adapter. L'obiettivo è quindi raccogliere gli eventi di aggiornamento (del processo di shadowing) in un unico flusso per poi spedirlo al Physical Asset Adapter.

Il Data Hub, come detto precedentemente, potrebbe essere sia un componente sviluppato dall'organizzazione sia un servizio in cloud che ha le stesse funzionalità, come ad esempio il servizio IoT Hub di Microsoft Azure. Inoltre, nel caso in cui sia sviluppato dall'organizzazione potrebbe essere sia un componente esterno al Physical Asset Adapter sia un sotto componente di

quest'ultimo. La natura e le funzionalità del componente non cambiano in quanto per come è progettato è dotato di una grande elasticità di deploy. Infatti, l'unico elemento che cambia è il canale di comunicazione tra Data Hub e Physical Asset Adapter. Nel caso in cloud sarà una web socket, nel caso in cui venga sviluppato dall'organizzazione rimanendo un componente esterno può essere sia una socket sia nuovamente una web socket e infine se fosse un sotto componente del Physical Asset Adapter si potrebbe utilizzare anche un event bus fornito dal framework a supporto di quest'ultimo.

In qualsiasi caso, il Physical Asset Adapter vedrà un unico flusso di eventi di shadowing espressi nello stesso formato. Quindi, non sarà in grado, e non deve esserlo, di riconoscere quale sia la sorgente dei dati. Esso si occuperà della ricezione dei dati dal Data Hub, della conversione di quest'ultimi al fine di adattarli al modello d'interesse e della richiesta di aggiornamento dei Digital Twin attraverso le API o gli SDK del Twin Builder utilizzato (nel caso del nodo in questione di Microsoft Azure Digital Twins), completando il processo di shadowing.

Parlando invece del Core, la cui visione di dettaglio è illustrata nella Figura 5.4, si può notare che il numero di componenti era già ben descritto a livello dell'architettura astratta dove però venivano descritti più in termini di requisiti che in termini effettivamente di deploy. Nella figura si può notare che alcuni componenti che precedentemente erano di primo livello, ora qui sono diventati sotto componenti.
L'elasticità dell'architettura consente di poter esprimere i componenti come si desidera, trasformando anche ciò che era un componente primario in un sotto componente se necessario. Come sottolineato sempre durante questa tesi l'aspetto fondamentale è il rispetto degli standard definiti a livello di ecosistema.

In questa versione proposta si vede il Relationship manager assieme a Digital Twins metadata diventare dei sotto componenti del Catalogo al fine di una gestione, a mio parere, migliore.

Come anche per la visione di dettaglio precedente, è possibile notare i collegamenti principali tra i vari componenti, utilizzati per poter collaborare e

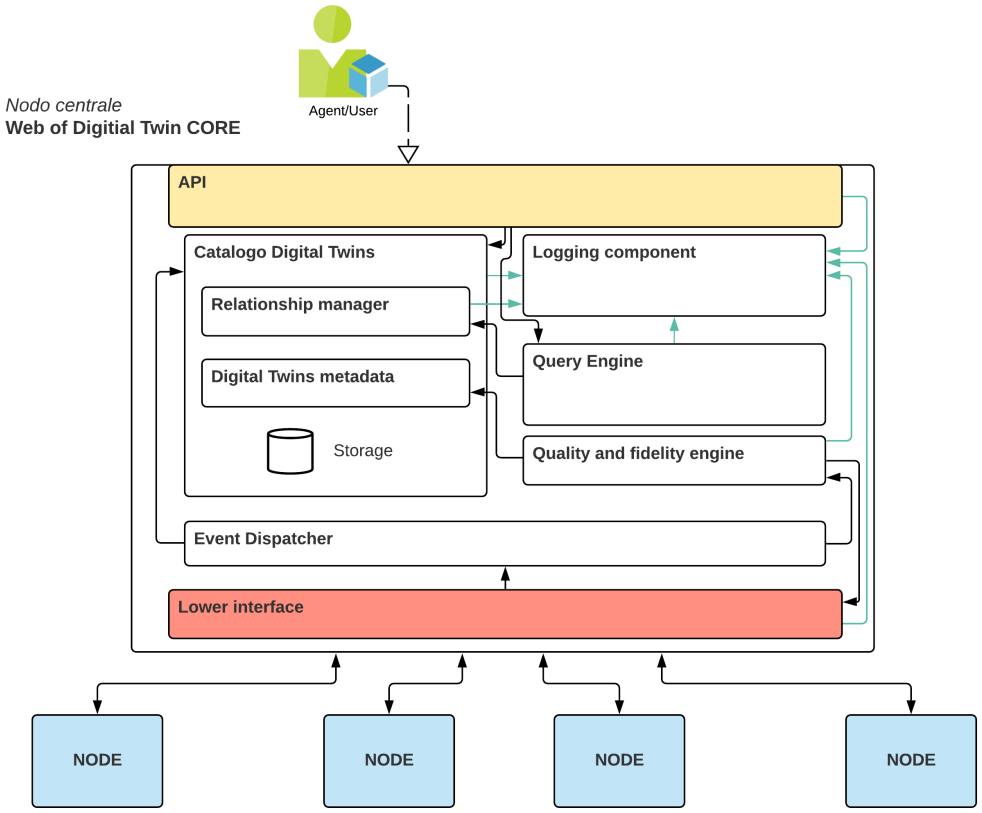

Figura 5.4: Visione di dettaglio della proposta di Core

raggiungere gli obiettivi stabiliti.

Per quanto riguarda la descrizione dei vari componenti essa è la medesima rispetto alla descrizione della visione astratta in quanto già ad un livello di dettaglio sufficiente per poter definire l'architettura.

L'unico aspetto da considerare con maggiore attenzione è la scelta di adottare una modellazione in RDF di tutte le informazioni di stato dei Digital Twins memorizzati all'interno del Catalogo. Il modello seguirà l'upper-ontology definita precedentemente e verrà memorizzata all'interno di un RDF Store. Il Catalogo andrà a gestire l'RDF Store in cui memorizzare tutti i riferimenti, assieme ai dati e ai metadati, dei Digital Twins presenti nell'ecosistema. Come visto, include dentro di sé anche i metadati e soprattutto la gestione del-

le relazioni inter-organizzazione. Questo perchè, adottando RDF e RDF Store, è possibile descrivere le relazioni tra Digital Twins semplicemente attraverso *statement* aggiuntivi all'interno dello stato di ogni Digital Twin descritto a livello del catalogo. In questo modo possiamo utilizzare tutti i vantaggi offerti dal linguaggio di query SPARQL e consentire agli agenti che si collegano di poter eseguire direttamente query in SPARQL senza la necessità di adottare convertitori ad-hoc come nel caso delle organizzazioni.

Inoltre, SPARQL può essere utilizzato anche dal Quality and Fidelity engine per poter fare delle query mirate ad individuare i Digital Twins che sono fuori sincronizzazione.

Quindi viene creato un ulteriore grafo di conoscenza che sta ad un livello più alto di tutti i grafi di conoscenza memorizzati all'interno delle singole organizzazioni.

Per quanto riguarda le API offerte dal Core, vengono individuate le principali:

- Query: consente agli agenti di collegarsi al Core per eseguire query sul grafo di conoscenza dell'ecosistema ottenendo dati e relazioni presenti. Ad esempio, si potrebbero eseguire query per ottenere tutti i Digital Twins appartenenti ad una specifica classe o tutti i Digital Twins di una specifica classe con particolari requisiti di fidelity. Questa funzionalità pone le basi per tante applicazioni e per il lavoro di tanti agenti, rivelandosi fondamentale in Web of Digital Twins. La richiesta viene specificata, come per il Web of Digital Twins Node, seguendo le recenti recommendation SPARQL. La richiesta è la seguente:

  POST http://dominiocore/api/twins/sparql

  La query, come da requisito, viene specificata in SPARQL. Inoltre, anche qui, in accordo con la recommendation W3C [2] utilizzo il metodo POST per la richiesta di esecuzione della query. Il body della richiesta contiene la query in SPARQL, mentre la risposta viene restituita in SPARQL-JSON [20]. Inoltre, gli *status code* che la richiesta deve restituire sono i seguenti:

  - 200 (OK): la query è stata processata correttamente e viene restituito il risultato.
  - 400 (Bad Request): la query ha degli errori e non può essere elaborata.
  - 500 (Internal Server Error): quando la richiesta fallisce a causa di errori lato server.

- Ottenimento dati Digital Twin: consente agli agenti di ottenere direttamente tutti i metadati di un Digital Twin. La richiesta è la seguente:

  GET http://dominiocore/api/twins?uri={uri del Digital Twin}

  L'URI del Digital Twin viene specificato come query string per il parametro "uri". Il risultato viene restituito in RDF. Inoltre, gli *status code* che la richiesta deve restituire sono i seguenti:

  - 200 (OK): risorsa trovata e restituita, il body contiene i metadati associati al Digital Twin.
  - 400 (Bad Request): richiesta mal formata, nel caso in cui non venga passato il parametro.
  - 404 (Not Found): Digital Twin non presente all'interno dell'ecosistema.
  - 500 (Internal Server Error): quando la richiesta fallisce a causa di errori lato server.

A queste ne possono essere aggiunte sicuramente altre come ad esempio un API per ottenere il mapping tra l'id di un'organizzazione e il suo URI ecc.. Anche qui, l'aspetto importante è sottolineare la necessità di un approccio formale per la definizione di ogni aspetto dell'architettura.

Un modo per comprendere meglio il flow dell'architettura è a mio parare descrivere un caso d'uso. Prendiamo in considerazione il caso in cui un agente abbia la necessità di osservare un particolare Digital Twin mentre esso viene

aggiornato e quindi durante il processo di shadowing. Esso ci consente di vedere le varie interazioni all'interno del Nodo e del Core.

Nella Figura 5.5 si può osservare tutto il processo con le corrispondenti etichette. Partendo dalla richiesta di osservazione, l'agente (icona verde indicata con "Agent/User") invia una richiesta "GET" all'endpoint (i1)

http://dominio/api/twins/{id}/events

Dopodiché, il componente API, analizzata la richiesta, converte la connessione in una web socket (i2) di cui se ne occuperà il sotto componente "Web Sockets". In questo momento l'agente è pronto a ricevere tutti gli eventi in cui il Digital Twin è coinvolto.
L'asset fisico aggiorna il proprio stato; supponiamo ci sia un cambio in una proprietà, perciò, l'asset, considerando sia fornito solo del protocollo MQTT, invia i nuovi dati sul topic "wodt/orgX/tipoDigitalTwin" per poter aggiornare la propria replica digitale (a1). L'MQTT Adapter, in ascolto sul topic "wodt/orgX/+", riceve i nuovi dati e provvede a spedirli al Data Hub, il quale raccoglie i dati da più sorgenti (a2). Il Data Hub esegue un controllo di validità sui dati appena ricevuti al fine di verificare la presenza di tutti i campi necessari. Una volta superato il controllo, invia i dati sull'unico stream di eventi che lo collega con il Physical Asset Adapter. Qui (a3) i dati vengono analizzati e convertiti nel modello adeguato al fine di poter richiamare, attraverso l'SDK di Microsoft (e quindi indirettamente attraverso le API), l'aggiornamento del Digital Twin. Una volta giunti all'istanza di Azure Digital Twins (a4) la piattaforma provvede ad aggiornare i dati dello stato di quel Digital Twin ed emette, su tutti gli endpoint che includono nel filtro eventi quella tipologia di evento, un evento di aggiornamento Digital Twin ("Microsoft.DigitalTwins.Twin.Update"). Gli endpoint configurati, quindi Event Grid per il processo di osservazione e Event Hubs per il processo di salvataggio della storia, raccolgono gli eventi e li inoltrano ai propri osservatori facendo scattare i trigger delle Azure Function associate (a5, a5-bis). Nel caso del processo per il threading (storia), l'Azure Function elabora l'evento, estrapola i dati importanti e li invia al sucessivo Event Hub che fa da sorgente per il Time

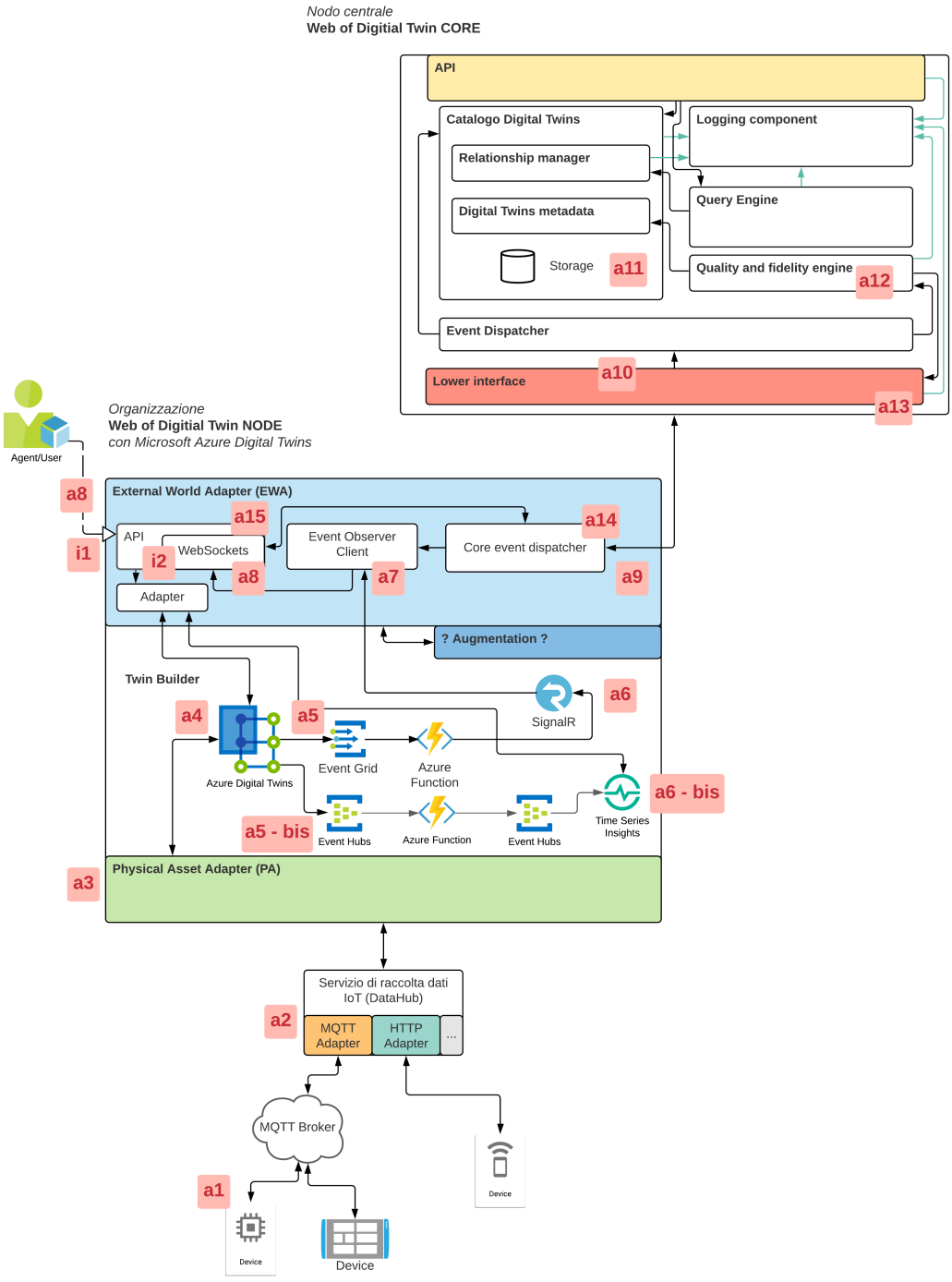

Figura 5.5: Flow per un processo che coinvolge osservazione, shadowing e controllo fidelity

Series Insights in cui i dati vengono elaborati e memorizzati (a6-bis). Invece, per il processo di tracking i dati giungono sempre ad una Azure Function, la quale, anche in questo caso, estrapola i dati importanti e ne aggiunge altri derivati dal payload dell'evento e li invia su un canale di SignalR. I dati, una volta giunti al servizio SignalR (a6), vengono inoltrati in modalità push a tutti gli ascoltatori, in particolare nel nostro caso all'Event Observer Client (a7) il quale riceve i dati, li elabora e li inoltra al sotto componente "Web Sockets" per poterli replicare e spedire a tutti gli agenti interessati. In questo caso, abbiamo il nostro agente che ha appena aperto una connessione per osservare quel Digital Twin quindi i nuovi dati verranno elaborati, rappresentati in un formato idoneo e spediti sulla web socket fino a raggiungere l'agente (a8) che poi potrà elaborarli e sfruttarli liberamente.
Allo stesso tempo, il Core Event Dispatcher riceve anch'esso l'evento di aggiornamento (attraverso il bus di eventi condiviso a livello di middleware) e provvede a notificare il Core generando un ulteriore evento da spedire attraverso il canale bidirezionale che li collega (a9). Il Core raccoglie l'evento attraverso la Lower Interface (a10) e lo spedisce all'Event Dispatcher che lo smista verso il "Catalogo" il quale aggiorna i metadati sulla data e ora di ultima modifica di quel Digital Twin (a11). Il Quality and Fidelity Engine si accorge dell'aggiornamento (a12) e, supponendo ci sia un cambiamento sullo stato di validità (cioè grazie all'aggiornamento il Digital Twin torna in uno stato di validità), crea un evento con cui informare il Nodo riguardo la validità del Digital Twin e lo spedisce attraverso la Lower Interface (a13) al Core Event Dispatcher dell'organizzazione (a14). A questo punto, il Core Event Dispatcher elabora l'evento e lo spedisce, attraverso il bus di eventi interno al middleware, all'API Component che aggiornerà i propri dati sulla validità dei Digital Twins all'interno del nodo (a15).

Volendo descrivere un ulteriore flow molto semplice, si potrebbe parlare della risoluzione di una query locale all'organizzazione. L'agente effettua una richiesta POST all'API al seguente indirizzo:

http://dominio/api/twins/sparql

specificando all'interno del body la query in SPARQL. La query, una volta giunta al componente API, viene passata all'Adapter il quale la analizza e se non vi sono errori, in tal caso restituirà un errore opportuno all'agente, la converte nell'SQL utilizzato da Azure Digital Twins al fine di poter richiedere l'esecuzione della query da parte del Twin Builder. A questo punto, viene effettivamente richiesta l'esecuzione della query ad Azure Digital Twins. Esso risponde inviando il risultato in formato JSON. Il risultato viene convertito dall'Adapter in SPARQL-JSON, aderendo quindi alle recommendation, e restituito all'agente tramite il componente API.

L'architettura proposta è indipendente dai linguaggi di programmazione e dalle tecnologie utilizzate. Ogni componente può essere sviluppato con gli strumenti che si preferiscono a patto che vengano rispettate le interfacce e gli "standard" definiti a livello di ecosistema. Addirittura, grazie all'approccio a microservizi, i singoli componenti di ogni nodo possono essere sviluppati con linguaggi di programmazione e con tecnologie completamente differenti.

Capitolo 6

Implementazione prototipale

In questo capitolo descriverò la parte implementativa che ha l'obiettivo di dimostrare la realizzabilità dell'architettura proposta. Nella prima parte verranno descritte le tecnologie utilizzate al fine di poter sviluppare le funzionalità richieste. Dopodiché, nella seconda e terza sezione verranno mostrate le parti principali dell'implementazione del Nodo e del Core.

6.1 Tecnologie utilizzate

Per lo sviluppo del progetto sono state utilizzate diverse tecnologie. Qui verranno descritte le principali al fine di creare un quadro tecnologico di base. Come anticipato, l'architettura proposta è completamente indipendente dai linguaggi di programmazione e dalle tecnologie utilizzate per la realizzazione.

L'obiettivo di questa sezione e in generale di questo capitolo è mostrare un esempio di implementazione solamente al fine di dimostrare l'applicabilità.

Il linguaggio di base su cui si basa la maggior parte del progetto è *Java*. Inoltre, al fine di avere a disposizione un ambiente reattivo che lavorasse ad eventi ho utilizzato il toolkit *Vert.x*. *Vert.x* ha concesso di implementare con più facilità alcune componenti o sotto componenti che utilizzano alcune tecnologie come le *Web Socket*, le *Socket TCP*, client *MQTT*, *web server* e ha fornito tutto il bus di eventi interno ai nodi e al core necessario per permettere il funzionamento event-driven.

Come da proposta, il Core lavora direttamente utilizzando *RDF*, quindi vi è stata la necessità di utilizzare RDF e conseguentemente un supporto di memorizzazione cioè un RDF Store. È stato utilizzato *TDB* offerto dal framework *Apache Jena* il quale è stato anche impiegato per la gestione completa dell'RDF.

Ovviamente, vista l'intenzione di sviluppare un nodo con i servizi Microsoft è stato necessario utilizzare tutte le tecnologie e i servizi indicati: *Azure Digital Twins, Event Grid, Event Bus, Azure Function, SignalR e Time Series Insights* assieme a tutti gli strumenti di management necessari per la loro gestione. Le parti di codice che riguardano i servizi Microsoft, in particolare le Azure Functions, sono state sviluppate in *C#*.

6.1.1 Java

Parte fondamentale per lo sviluppo dei principali componenti dell'architettura è Java. Java è un linguaggio di programmazione ad alto livello, orientato agli oggetti e a tipizzazione statica[1].
La versione di Java utilizzata in questo progetto è la 11, uscita a fine 2018.
Il linguaggio fu inventato da *James Gosling* e la prima versione fu annunciata pubblicamente nel 1995 a SunWorld. Da allora il linguaggio ha subito numerose modifiche ed integrazioni che lo hanno portato ad essere uno dei linguaggi di programmazione più utilizzati al mondo. Ora è gestito commercialmente da Oracle, anche se sono tutt'ora presenti versioni open, come quella utilizzata durante lo sviluppo di questo progetto (OpenJDK). Uno dei punti di forza che lo ha portato ad essere uno dei linguaggi di programmazione più utilizzati al mondo è il principio *WORA*(*write once, run anywhere*). Infatti, il codice viene compilato in un formato chiamato *bytecode* che viene eseguito su una macchina virtuale Java chiamata *Java Virtual Machine*. La compatibilità è data dal fatto che la Java Virtual Machine è disponibile per tantissime piattaforme e sistemi operativi e il codice, eseguendo appunto su di essa, non deve essere ricompilato in bytecode ogni volta.

[1] `https://it.wikipedia.org/wiki/Java_(linguaggio_di_programmazione)`

6.1.2 Vert.x

Eclipse Vert.x è un toolkit che consente di creare applicazioni reattive sulla JVM(Java Virtual Machine). La *programmazione reattiva* è un paradigma di programmazione associata a stream asincroni con la possibilità di rispondere ad ogni cambiamento e ad ogni evento. Vert.x è completamente event-driven, non-blocking e riesce a gestire tantissimi servizi in concorrenza utilizzando pochissimi thread. Utilizza un event bus per consentire la comunicazione tra le diverse parti dell'applicazione e quindi per passare gli eventi in modo asincrono ai vari handler definiti. Un componente di Vert.x si chiama *verticle* e consente, attraverso un singolo thread, di gestire un event loop con il quale mettere a disposizione una quantità enorme di servizi. Un'applicazione tipicamente è composta da più verticles che eseguono sulla stessa istanza di Vert.x (ma è possibile farli eseguire anche su istanze diverse) e comunicano tra di loro con un approccio event-driven attraverso l'event bus (anche nel caso di più istanze). I verticle possono mettere a disposizione una quantità di servizi pressoché illimitati. Infatti, possiamo creare socket, web socket, web server, mqtt broker, mqtt client, server mail, gestire database e tanto altro.
Vert.x è leggero, veloce, modulare, ideale per i microservizi e poliglotta(grazie al fatto che lavora sulla JVM e quindi supporta tutti i linguaggi sviluppati sopra ad essa).

All'interno di questo progetto viene utilizzato per la creazione di socket, web socket, web service RESTful, client MQTT e tanto altro. La versione utilizzata è la 4.

6.1.3 Socket e Web Socket

Una socket è un oggetto software che consente lo scambio di dati tra host remoti o nel caso IPC anche tra processi locali. Una socket è individuata attraverso un indirizzo IP e un numero di porta. All'interno del progetto viene utilizzata una Stream socket (socket TCP) per il collegamento tra Nodo e Core. Le Stream Socket sono basate sul protocollo TCP e garantiscono una comunicazione affidabile, full-duplex e orientata alla connessione.

Le Web Socket sono una tecnologia che consentono la creazione di un canale di comunicazione bidirezionale attraverso il quale un server può spedire eventi ad un client (e viceversa) che li riceve senza la necessità di eseguire un polling continuo al server. Quindi, questa tecnologia facilita le comunicazioni in real-time tra client e server. Il protocollo (standardizzato dal W3C) è un'implementazione basata sul protocollo TCP. È correlato con l'HTTP nel modo in cui effettua l'handshake. Infatti, oltre ad utilizzare le stesse porte dell'HTTP, le Web Socket utilizzano, durante l'handshake, l'header HTTP "Upgrade header" per passare dal protocollo HTTP (utilizzato per la richiesta iniziale) al protocollo WebSocket.
Esse sono utilizzate all'interno del progetto per la creazione del canale attraverso cui l'agente può osservare un Digital Twin e ricevere tutti gli eventi in cui esso è coinvolto. Viene creata grazie alla possibilità di conversione di una richiesta HTTP in una Web Socket offerta dal toolkit Vert.x.

6.1.4 MQTT

MQTT(Message Queuing Telemetry Transport) è un protocollo di messaggistica per IoT. È un protocollo molto leggero di tipo pubblish/subscribe ed è, per questo motivo, ideale per la ricezione di dati anche da piccoli device e sensori in zone remote. Il pattern pubblish/subscribe implementato in MQTT necessita di un *broker* intermedio per il funzionamento e per la distribuzione dei messaggi ai client. Esso si basa sul protocollo di trasporto TCP e permette diversi livelli di QoS al fine di prevedere connessioni anche molto robuste (at most once, at least once, exactly once). Il funzionamento è il seguente: i client che desiderano inviare un messaggio pubblicano i dati su un topic all'interno del message broker e ogni client interessato in quel tipo di dato effettua la subscribe a quel topic in modo da ricevere i messaggi in modo asincrono, cioè senza la necessità di eseguire un polling al broker. Infatti, una peculiarità di questa tecnologia è che non appena è disponibile un messaggio su un topic, il message broker lo replica su tutte le sue sottoscrizioni assicurando un certo livello di affidabilità nella comunicazione a seconda del QoS scelto per la con-

nessione. Perciò, gli elementi di base che coinvolgono MQTT sono: un client che spedisce i dati, un broker che gestisce le comunicazioni e un ulteriore client che effettua la subscribe.
All'interno del progetto è stato utilizzato per fornire ai device uno dei metodi per spedire i dati al Data Hub per il processo di shadowing. Al fine di utilizzare il protocollo tramite semplici API è stata utilizzata la libreria *Eclipse Paho* per la creazione di un semplice device che inviasse i dati per il processo di shadowing ed un client MQTT offerto da Vert.x per la creazione del client presente all'interno dell'MQTT Adapter (modulo del Data Hub) per la raccolta dati (attraverso la subscribe ai topic descritti precedentemente). Inoltre, è stato utilizzato *Eclipse Moquitto*, un message broker open-source che implementa le versioni 5, 3.1.1 e 3.1 del protocollo MQTT.

6.1.5 RDF e RDF Store TDB

RDF(Resource Description Framework) è un modello proposto dal W3C per la codifica, lo scambio e il riutilizzo di dati assicurando l'interoperabilità semantica tra applicazioni all'interno del Web. RDF estende il sistema di linking presente nel Web attraverso l'uso di URI anche per la definizione delle relazioni che legano le risorse che si trovano sul Web Semantico (identificate a loro volta tramite URI). Il modello di base prevede che le informazioni vengano memorizzate in *triple*. Ciascuna tripla, definita *statement*, è composta da, appunto, tre elementi: soggetto, predicato e oggetto. Il soggetto è la risorsa a cui l'informazione si riferisce, il predicato è la relazione che lega il soggetto con l'oggetto. L'oggetto può essere sia un ulteriore risorsa sia un semplice valore (stringa, numero o qualsiasi tipo di dato supportato). I predicati sono identificati da URI così come le risorse al fine di assicurare il riferimento al suo significato esatto. Ad esempio, potremmo avere due relazioni che hanno lo stesso nome ma, appartenendo a due domini diversi, potrebbero avere un significato completamente diverso. Identificando un predicato attraverso il suo URI, riusciamo a collegare il nome di quella relazione al suo significato e questo è un passo fondamentale nella realizzazione di un Web Semantico.

Questa struttura, in cui le informazioni e le risorse sono legate tra di loro attraverso dei link, va a formare un grafo che viene detto *grafo di conoscenza*. Al fine di memorizzare le triple, e quindi il grafo, si potrebbero utilizzare i database tradizionali anche se, solitamente, vengono impiegati dei database costruiti ad-hoc per la gestione efficiente di un grafo di conoscenza. Questi vengono detti RDF Store.
Come detto prima, il Web Semantico ha tanti aspetti in comune con Web of Digital Twins, perciò RDF viene utilizzato come linguaggio di descrizione dei dati. Al fine di lavorare con l'RDF e poterne eseguire la memorizzazione è stato utilizzato un framework chiamato *Apache Jena* che consente di costruire applicazioni che si basano su RDF. Inoltre, Apache Jena offre un RDF Store chiamato *TDB*. TDB è un componente di Jena per lo storage e per l'esecuzione di query (in SPARQL). TDB è un RDF Store ad alte performance su singola JVM, quindi adatto al contesto dimostrativo in questione.

6.1.6 SPARQL

SPARQL è un linguaggio di interrogazione per i dati rappresentati in RDF che consente di estrarre le informazioni da un grafo di conoscenza o in generale da un RDF Store. Una semplice query è composta da due parti principali: la clausola "SELECT" che identifica le variabili che devono apparire nel risultato della query, e la clausola "WHERE" che contiene il pattern della query rispetto al grafo di conoscenza. Il pattern è espresso sempre in triple in cui le variabili che non si conoscono vengono indicate come "question word" e possono apparire in più statement (anche al fine di legarli per ottenere informazioni derivate da relazioni tra più risorse). Oltre alle triple, possono comparire altri elementi come ad esempio gli "OPTIONAL" o i "FILTER".
All'interno del progetto, come da requisito, viene utilizzato per specificare le query all'interno delle API opportune.

6.1.7 DTDL

I modelli dei Digital Twins in Azure Digital Twins vengono definiti tramite il linguaggio *DTDL*(Digital Twins Definition Language) v.2. Esso è basato su JSON-LD ed è indipendente dal linguaggio di programmazione. Al fine di validare i modelli ho utilizzato anche uno strumento creato da Microsoft chiamato *DTDL Validator*.

6.1.8 C#

Utilizzato per la parte dei servizi Microsoft, in particolare per le Azure Functions, *C#* è un linguaggio di programmazione orientato agli oggetti, anche se con le ultime versioni è definibile come linguaggio multi-paradigma. Il linguaggio è stato progettato da *Anders Hejlsberg* all'interno di *Microsoft* come componente dell'iniziativa *.NET*. La prima versione fu rilasciata nel 2002 e da allora ha subito numerose modifiche ed integrazioni che lo hanno portato ad essere nelle classifiche dei linguaggi più utilizzati nel mondo.
Il linguaggio, come Java, subisce una compilazione intermedia producendo, appunto, codice intermedio denominato CIL. Dopodiché, un componente del *.NET Framework* chiamato *CLR*(Common Language Runtime), il quale è associabile alla JVM per Java, carica il codice CIL e ne effettua una compilazione JIT(Just in Time) ai fini dell'esecuzione.

6.1.9 Servizi Microsoft e strumenti di gestione

Come anticipato nella proposta di architettura, si vuole realizzare un nodo che utilizzi la soluzione di Microsoft per i Digital Twins. È necessario utilizzare i seguenti servizi (e le rispettive API), descritti durante la proposta di architettura: *Azure Digital Twins, Event Grid, Event Hub, Azure Function, SignalR e Time Series Insights*.
Per la loro creazione e la loro gestione ho utilizzato il servizio *Microsoft Azure Portal* in combinazione con la *Powershell*, grazie all'estensione *azure* sviluppata sempre da Microsoft. Attraverso Microsoft Azure Portal è possibile gestire

e visualizzare tutti i servizi Azure utilizzati e le proprie sottoscrizioni. Inoltre, un ulteriore strumento utilizzato per visualizzare il grafo di conoscenza (creato da Azure Digital Twins) è stato *Azure Digital Twins Explorer* che mette a disposizione anche una serie di funzionalità utili per la gestione del grafo stesso.

6.2 Implementazione

Dopo aver descritto le tecnologie principali impiegate all'interno del progetto è opportuno descrivere la fase di implementazione. Nella prima parte di questa sezione verrà descritto il processo di deploy di tutti i componenti dell'architettura a partire dai servizi Microsoft fino ai componenti Node e Core dell'architettura proposta. Dopodiché, verranno descritti i punti di maggiore importanza all'interno delle implementazioni del Node e del Core.

6.2.1 Deploy

La parte principale del deploy riguarda i servizi Microsoft. I vari servizi utilizzati sono tutti servizi in cloud e quindi è necessario utilizzare gli strumenti offerti da Microsoft per la loro gestione e il loro deploy. Gli strumenti coinvolti sono, come anticipato nella parte delle tecnologie, Microsoft Azure Portal e la Powershell grazie all'estensione per Azure.
È possibile notare lo schema di deploy del Twin Builder, interno al nodo da costruire, nella Figura 6.1. Qui, verrà illustrato il processo di deploy che coinvolge la parte di tracking e quindi da Azure Digital Twins fino a SignalR. Non verrà mostrata la parte fino a Time Series Insights in quanto il processo è veramente molto simile e quindi evito ripetizioni.

Si considera qui che la Subscription per poter utilizzare i servizi di Microsoft Azure sia già stata creata.

Il primo passo è quello di creare un *resource group*, cioè un gruppo di risorse. Un gruppo di risorse in Azure è un contenitore con risorse correlate per una soluzione Azure. Il gruppo di risorse include solitamente tutte le

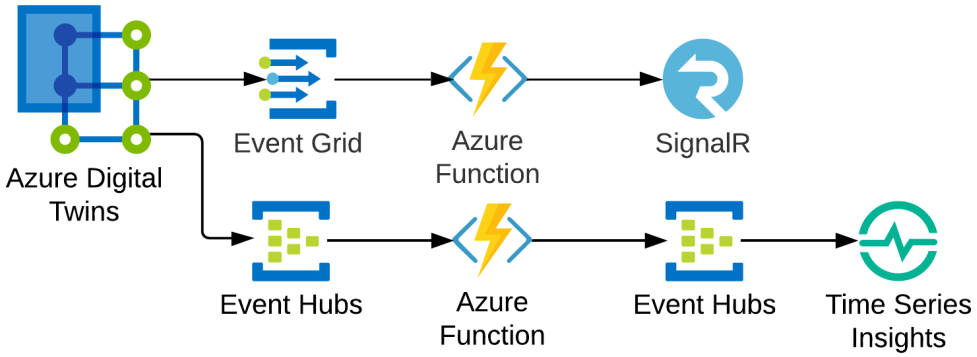

Figura 6.1: Servizi Microsoft

risorse coinvolte all'interno di uno stesso progetto o di una stessa soluzione e quindi si consiglia di inserire quelle risorse che condivideranno lo stesso ciclo di vita. In questo caso è possibile creare un unico gruppo di risorse per tutti i servizi coinvolti nel singolo Nodo. Per effettuare la creazione è necessario accedere ad Azure Portal e crearlo attraverso la GUI offerta. I dati da inserire riguardano la sottoscrizione per la quale desideriamo crearlo, la regione in cui memorizzarlo ed infine il nome. Il nome dato al resource group, in questo caso (sperimentale per la tesi), è *thesis-middlwareforwodt* e si vedrà comparire anche nei comandi descritti successivamente.

Dopo aver creato il gruppo di risorse è possibile iniziare a creare le istanze dei vari servizi da utilizzare. Si può iniziare da Azure Digital Twins e la sua creazione, come quella di molti servizi in questa architettura, può essere eseguita semplicemente da Azure Portal. Quindi, ci si reca all'interno del gruppo di risorse appena creato attraverso il Portal e si procede alla creazione del servizio. È necessario anche qui, come in tutti i servizi, la subscription, il resource group, la regione e il nome del servizio. Al servizio Azure Digital Twins, per il nodo di prova, assegno il nome *Org1-TwinBuilder*. Dopo aver creato il servizio principale, è possibile procedere al deploy di tutti i servizi necessari per il tracking di un Digital Twin. L'obiettivo è la creazione di un endpoint (Event Grid) a cui spedire tutti gli eventi di Azure Digital Twins (con

la corrispettiva event route) e il successivo collegamento all'Azure Function in modo che possa elaborare i dati e spedirli a SignalR che dovrà fornire il servizio di comunicazione in real-time. Innanzitutto, è necessario creare l'Event Grid Topic, interno al resource group, a cui Azure Digital Twins invierà i dati e a cui l'Azure Function verrà registrata per configurare il trigger. La sua creazione può essere effettuata dal Portal, ma in questo caso preferisco utilizzare la Powershell. La creazione richiede l'esecuzione del seguente comando:

```
az eventgrid topic create --resource-group $resgroup --name
    "adteventstopic" --location "northeurope"
```

Nei comandi si considerino le variabili *$resgroup* e *$adt* come contenenti rispettivamente i nomi del gruppo di risorse e dell'istanza di Azure Digital Twins creati precedentemente. Dopodiché, è necessario creare l'endpoint su Azure Digital Twins in modo tale da creare il collegamento (ancora senza route per passare gli eventi) con il topic appena creato:

```
az dt endpoint create eventgrid --dt-name $adt
    --eventgrid-resource-group $resgroup --eventgrid-topic
    "adteventstopic" --endpoint-name "adteventgridendpoint"
```

In output si otterranno in entrambi i casi dei risultati in formato JSON in cui si può osservare la riuscita o meno dei comandi.
Come definito nella descrizione delle tecnologie, è necessario, una volta collegato un endpoint, definire una event route che consenta agli eventi di essere smistati verso gli endpoint con gli opportuni filtri (opzionali):

```
az dt route create --dt-name $adt --endpoint-name
    "adteventgridendpoint" --route-name "adteventgridroute"
```

Con questo comando verrà creata una event route con filtro impostato a *true* in modo tale da ricevere qualsiasi tipologia di evento (come richiesto dal processo di osservazione).

Ora è necessario, prima del deploy dell'Azure Function, definire l'altro estremo del processo e quindi creare l'istanza del servizio SignalR. In questo

caso è molto più semplice la creazione attravero l'Azure Portal in cui inseriremo i soliti dati più il *pricing tier* che, per l'obiettivo della tesi, impostiamo a *free* e la modilità di servizio che impostiamo a *Serveless*. Questa modalità è necessaria per poter lavorare con un'architettura di questo tipo in cui l'endpoint per la fase di negoziazione (per l'ottenimento del token di accesso e dell'url del servizio) è offerto da una Azure Function. Infatti, poiché i due servizi (Azure SignalR ed Azure Function) sono servizi completamente gestiti e altamente scalabili è comune utilizzarli assieme per fornire comunicazioni in tempo reale in un ambiente senza server che le gestisca.

A questo punto occorre creare tutto il necessario per la creazione dell'Azure Function che collega i due servizi. Innanzitutto, occorre creare un account per lo storage che verrà utilizzato dalla funzione:

```
az storage account create --name "adtstoragefunctionapp"
    --resource-group $resgroup --location northeurope
```

Dopodiché, occorre creare il "contenitore" per le Azure Function, chiamato *Azure Function App*:

```
az functionapp create --resource-group $resgroup
    --consumption-plan-location "northeurope" --runtime dotnet
    --functions-version 3 --name "eventgridsignalrfunctionapp"
    --storage-account "adtstoragefunctionapp"
```

Qui viene specificato anche il linguaggio utilizzato all'interno delle funzioni (C#) e l'account per lo storage appena creato.
Dopo aver soddisfatto tutti i pre requisiti, è possibile procedere alla creazione e al deploy dell'Azure Function. Questo può essere fatto interamente tramite *Visual Studio Code*. Il codice inserito all'interno dell'Azure Function è il seguente:

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using Microsoft.Azure.WebJobs.Extensions.EventGrid;
using Microsoft.Azure.EventGrid.Models;
using Microsoft.Azure.WebJobs;
using Microsoft.Extensions.Logging;
using Microsoft.Azure.WebJobs.Extensions.Http;
using Microsoft.AspNetCore.Http;
using Microsoft.Azure.WebJobs.Extensions.SignalRService;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

namespace SignalRFunctions
{
    public static class SignalRFunctions
    {
        [FunctionName("negotiate")]
        public static SignalRConnectionInfo GetSignalRInfo(
            [HttpTrigger(AuthorizationLevel.Anonymous, "post")]
                HttpRequest req,
            [SignalRConnectionInfo(HubName = "dttelemetry")]
                SignalRConnectionInfo connectionInfo)
        {
            return connectionInfo;
        }

        [FunctionName("broadcast")]
        public static Task SendMessage(
            [EventGridTrigger] EventGridEvent eventGridEvent,
            [SignalR(HubName = "dttelemetry", ConnectionStringSetting
                = "AzureSignalRConnectionString")]
                IAsyncCollector<SignalRMessage> signalRMessages,
            ILogger log)
        {
```

```
            JObject message = (JObject)JsonConvert.DeserializeObject(
                eventGridEvent.Data.ToString());
            message.Add("Id", eventGridEvent.Subject);
            message.Add("EventType", eventGridEvent.EventType);
            message.Add("EventDateTime",
                eventGridEvent.EventTime.ToString());
            log.LogInformation($"Reading event:
                {message.ToString()}");

            return signalRMessages.AddAsync(
                new SignalRMessage
                {
                    Target = "newMessage",
                    Arguments = new[] {
                        JsonConvert.SerializeObject(message)
                    }
                });
        }

    }
}
```

Listato 6.1: Codice Azure Function tra Event Grid e SignalR

Come si può notare, sono presenti due funzioni. La prima è la funzione di negoziazione che serve ai client (nel nostro caso l'EventObserverClient) per connettersi al servizio SignalR e quindi ottenere le informazioni necessarie. La seconda invece, è quella che viene azionata attraverso il trigger dall'Event Grid non appena si verifica un evento. Essa elabora l'evento, aggiunge le informazioni necessarie e lo spedisce a SignalR attraverso l'output binding. Il trigger e l'output binding vengono definiti come parametri della funzione opportunamente annotati. Nell'annotazione per l'output binding di SignalR è possibile notare il campo "ConnectionStringSetting", che rappresenta la stringa di connessione al servizio, associato all'alias al quale riferirsi per ottenere

il dato dall'Azure Function App. Infatti, l'Azure Function App consente di definire delle costanti e memorizzarle all'interno dell'istanza del servizio stesso attraverso degli alias che poi possono essere utilizzati nelle funzioni per riferirsi a quel valore. Quindi, una volta definita la funzione ed eseguito il deploy attraverso Visual Studio Code, è necessario definire l'alias appena utilizzato e memorizzarlo nella Azure Function App:

```
az functionapp config appsettings set --resource-group $resgroup
    --name "eventgridsignalrfunctionapp" --settings
    "AzureSignalRConnectionString=$connString"
```

La variabile *$connString* contiene la connection string in questione, non inserita all'interno della tesi per motivi di spazio.
L'unico aspetto che rimane da definire rispetto alla Azure Function è il trigger dell'Event Grid che la fa azionare. Per fare ciò è sufficiente, tramite l'Azure Portal, creare un nuovo subscriber al topic ed impostarlo alla funzione "broadcast" appena creata.

Il deploy della parte che coinvolge i servizi per supportare la storia dei Digital Twins è molto simile al precedente e quindi, per evitare ripetizioni, non la descriverò nel dettaglio.

Dopodiché, è necessario eseguire il deploy della restante parte dell'architettura. I Nodi e il Core sono software scritti in Java sfruttando il toolkit Vert.x perciò il loro deploy, a livello dimostrativo, è semplice e corrisponde alla creazione dei verticle necessari e all'avvio del software. Per quanto riguarda invece il broker MQTT ho optato per il broker Mosquitto[2]. Anche il deploy di questo servizio è molto semplice e diretto attraverso una shell.

L'ultimo aspetto di interesse è la necessità dei componenti *Physical Asset Adapter* ed *External World Adapter*, interni al Nodo, di avere i permessi necessari per accedere alle API di Azure Digital Twins. Per fare ciò è necessario creare all'interno del servizio due AAD App (Azure Active Directory App) al fine di ottenere le credenziali per autenticarsi all'SDK. Occorre evitare che

[2] `https://github.com/eclipse/mosquitto`

tutti abbiano il potere di fare qualsiasi azione, perciò è opportuno definire in modo scrupoloso i permessi associati al PA e all'EWA. Azure Digital Twins definisce due tipologie di permesso:

- Azure Digital Twins Data Reader: solo permessi di lettura.
- Azure Digital Twins Data Owner: permessi completi.

Tenendo in considerazione che il Physical Asset Adapter deve solo eseguire operazioni di scrittura e in Azure Digital Twins non esiste un permesso preregistrato, ho creato un permesso ad-hoc per il contesto: *Azure Digital Twins Data Writer*. Esso è dotato del solo permesso di scrittura sull'istanza e deve essere definito a livello di *Azure Access Control*(la sezione in cui vengono definiti e gestiti i permessi interni alla subscription). Invece, l'EWA deve eseguire sia operazioni in scrittura sia in lettura perciò il permesso più appropriato è quello di *Data Owner*.
Dopo aver deciso i permessi da assegnare è necessario, attraverso l'Azure Portal o tramite Power Shell, procedere alla creazione delle due Azure Active Directory App per ottenere le credenziali di accesso. I dati necessari per l'autenticazione sono:

- Id dell'app: è l'id assegnato all'app creata.
- Password: è il token segreto per eseguire l'autenticazione.
- Tenant: è l'identificativo del tenant e quindi dell'organizzazione sotto cui la nostra subscription è definita. È uguale per tutte le app create all'interno della stessa organizzazione.

Questi dati verranno forniti all'SDK per eseguire l'autenticazione e ci consentiranno di poter utilizzare i servizi di Azure Digital Twins direttamente all'interno del codice Java definito per il Nodo.

6.2.2 Implementazione Nodo

Il middleware per il nodo è stato sviluppato in Java con il supporto del toolkit Vert.x. La progettazione dei verticles da utilizzare è di fondamentale im-

portanza. I verticles rispecchieranno quelli che sono i microservizi offerti nello schema architetturale e quindi i vari componenti. Una descrizione schematica si trova in Figura 6.2. Dopodiché, procederò nella descrizione e mostrerò le parti di codice principali (fornire l'intero codice sviluppato occuperebbe troppo spazio all'interno della tesi).

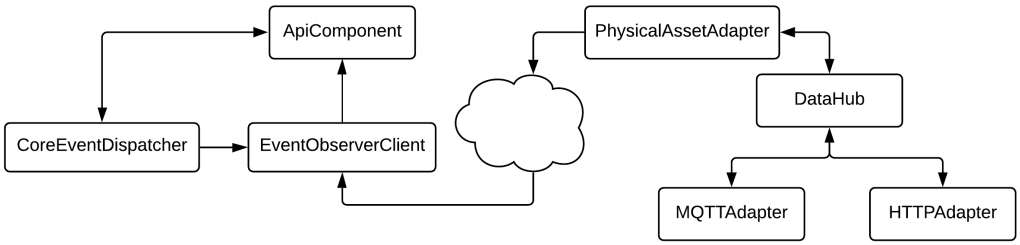

Figura 6.2: Verticles all'interno del middleware per il Nodo

Il DataHub è il verticle che si occupa del deploy dei vari moduli che fungono da adapter per le varie tecnologie supportate nel processo di shadowing. Inoltre, come anticipato, colleziona tutti i dati provenienti dai moduli, effettua i controlli di validità e spedisce i dati al Physical Asset Adapter per la parte finale del processo di shadowing. La parte principale del codice è la seguente:

```
package verticles.physicaladapter.datahubadapter;

import io.vertx.core.AbstractVerticle;
import io.vertx.core.eventbus.EventBus;
import io.vertx.core.json.JsonObject;
import verticles.physicaladapter.PhysicalAssetAdapter;

public class DataHub extends AbstractVerticle {
    /**
     * Event Bus Address for new incoming data.
     */
    protected static final String DATA_HUB_NEW_DATA =
        "datahub.newdata";
    /**
```

```
     * Event Bus Address for Digital Twins relationships management.
     */
    protected static final String DATA_HUB_RELATIONSHIPS =
        "datahub.relationships";
    /**
     * Label for specify Digital Twin type in the message data body.
     */
    public static final String DATA_HUB_NEW_DATA_DT_TYPE_LABEL =
        "DtType";
    /**
     * Label for specify Digital Twin ID in the message data body.
     */
    public static final String DATA_HUB_NEW_DATA_DT_ID = "DtId";
    /**
     * Label for specify Digital Twin target ID in relationships in
         the message data body.
     */
    public static final String DATA_HUB_NEW_DATA_DT_TARGET =
        "target";
    /**
     * Label for specify Digital Twin relationship name in the
         message data body.
     */
    public static final String DATA_HUB_NEW_DATA_DT_REL_NAME =
        "relation";

    @Override
    public final void start() {
        final EventBus eventBus = vertx.eventBus();
        //Deploy all the adapter verticles
        vertx.deployVerticle(new MQTTAdapter());
        vertx.deployVerticle(new HTTPAdapter());

        //Listen to incoming adapter's data. All the source will send
```

```
        data on this address.
    eventBus.consumer(DataHub.DATA_HUB_NEW_DATA, msg -> {
        //check that there is the id of the twin and the type
        final JsonObject jsonObj = new
            JsonObject(msg.body().toString());
        if (this.checkValidity(jsonObj)) {
            //Message validated
            //Send data to the Physical Asset Adapter
            eventBus.request(
                PhysicalAssetAdapter.PA_SHADOWING_NEW_DATA,
                jsonObj, ar -> {
                if (ar.succeeded()) {
                    //Send Physical Asset Adapter request's
                        results to the adapter.
                    msg.reply(ar.result().body());
                }
            });
        } //else nothing, message discarded.
    });

    //Listen to incoming adapter's data on relationships. All the
        source will send data on this address.
    eventBus.consumer(DataHub.DATA_HUB_RELATIONSHIPS, msg -> {
        final JsonObject jsonObj = new
            JsonObject(msg.body().toString());
        //check message validity
        if (this.checkRelValidity(jsonObj)) {
            //Message validated
            //Send data to the Physical Asset Adapter
            eventBus.request(
                PhysicalAssetAdapter.PA_SHADOWING_RELATIONSHIPS,
                jsonObj, ar -> {
                if (ar.succeeded()) {
                    //Send Physical Asset Adapter request's
```

```
                        results to the adapter.
                    msg.reply(ar.result().body());
                }
            });
        } //else nothing, message discarded.
    });
}

private boolean checkValidity(final JsonObject jsonObj) {
    //Method created for extendibility purposes.
    return jsonObj.containsKey(
        DataHub.DATA_HUB_NEW_DATA_DT_TYPE_LABEL)
            && jsonObj.containsKey(
                DataHub.DATA_HUB_NEW_DATA_DT_ID);
}

private boolean checkRelValidity(final JsonObject jsonObj) {
    //Method created for extendibility purposes.
    return jsonObj.containsKey(
        DataHub.DATA_HUB_NEW_DATA_DT_REL_NAME)
            && jsonObj.containsKey(
                DataHub.DATA_HUB_NEW_DATA_DT_TARGET)
            && jsonObj.containsKey(
                DataHub.DATA_HUB_NEW_DATA_DT_ID);
}
}
```

Listato 6.2: Codice DataHub

Questa è la parte principale di DataHub. Si può notare il rispetto dei requisiti dell'architettura. Il DataHub riceve gli eventi attraverso l'event bus di Vert.x dai due adapter implementati (per ora): un adapter MQTT e uno HTTP. Dopodichè, effettua un controllo di validità che, se superato, porta i dati ad essere inviati al Physical Asset Adapter attraverso l'event bus e quindi

come un unico stream di eventi espressi in un formato uniforme e indipendente dalla sorgente. Un esempio di adapter per il dialogo con i device che implementa solo l'update dei dati (relazioni tralasciate qui per motivi di spazio) è il seguente:

```
package verticles.physicaladapter.datahubadapter;

import io.vertx.core.AbstractVerticle;
import io.vertx.core.Future;
import io.vertx.core.eventbus.EventBus;
import io.vertx.core.json.JsonObject;
import io.vertx.mqtt.MqttClient;
import io.vertx.mqtt.messages.MqttConnAckMessage;

public class MQTTAdapter extends AbstractVerticle {

    private static final int MQTT_PORT = 1883;
    private static final String ROOT_TOPIC = "wodt/org1/+";
    private static final String HOST_NAME = "localhost";

    @Override
    public final void start() {
        final EventBus eventBus = vertx.eventBus();
        final MqttClient client = MqttClient.create(vertx);
        //Connection to broker, for educational purpose in local
            (it's a Mosquitto broker)
        Future<MqttConnAckMessage> future = client.connect(
            MQTTAdapter.MQTT_PORT, MQTTAdapter.HOST_NAME);
        future.onComplete((ar) -> {
            if (ar.succeeded()) {
                client.publishHandler((msg) -> {
                    final String dtType =
                        msg.topicName().split("/")[2];
                    final JsonObject msgPayload =
```

```
                    msg.payload().toJsonObject();
                eventBus.send( DataHub.DATA_HUB_NEW_DATA,
                    msgPayload.put(
                    DataHub.DATA_HUB_NEW_DATA_DT_TYPE_LABEL,
                    dtType));
            }).subscribe(MQTTAdapter.ROOT_TOPIC, 0);
        }
    });
  }
}
```

Listato 6.3: Codice principale MQTTAdapter

Come si può notare viene creato un client MQTT attraverso il toolkit Vert.x. Il client rimane in ascolto sul topic *wodt/org1/+* utilizzando la wildcard per includere tutti i topic di un livello superiore. Inoltre, nel topic è presente *org1* che corrisponde all'id dell'organizzazione creata (e a cui corrisponde il nodo). Non appena arriva un messaggio viene estratto il tipo di asset dal topic (il nome al posto del +) e vengono inviati i dati al DataHub, senza eseguire alcun controllo, come da requisiti. La restante parte sulle relazioni e l'HTTPAdapter, come qualsiasi altro adapter sviluppato, seguono lo stesso principio.
Questa struttura consente di poter sviluppare adapter per le varie tecnologie e aggiungerli senza modificare il codice del DataHub (tranne per la parte di deploy) rendendo tutto il sistema facilmente scalabile a fronte di nuove esigenze.

Il PhysicalAssetAdapter invece, è il verticle che si occupa di ottenere i dati dal DataHub, convertirli rispetto al modello di Digital Twin desiderato e provvedere all'aggiornamento del Digital Twin interno al Twin Builder sfruttando l'SDK offerto da quest'ultimo.
La parte principale del codice, è la seguente:

```
private TwinBuilderAsyncShadower twinBuilderShadower;

@Override
```

```
public final void start() {
    final EventBus eventBus = vertx.eventBus();
    //Create a promise in order to understand when the connection
       request is satisfied.
    final Promise<Void> isConnectedToADT = Promise.promise();

    this.twinBuilderShadower = new ADTAsyncCrud();

    vertx.executeBlocking(promise -> {
        try {
            //Connect to Azure Digital Twins instance
            this.twinBuilderShadower.connect(
                PhysicalAssetAdapter.TENANT_ID,
                PhysicalAssetAdapter.APP_ID,
                PhysicalAssetAdapter.APP_SECRET,
                PhysicalAssetAdapter.HOSTNAME);
            promise.complete();
        } catch (Exception e) {
            promise.fail(e);
        }
    }, res -> {
        if (res.succeeded()) {
            isConnectedToADT.complete();
        } else {
            System.out.println("(PA) Problems in connecting to Azure
                Digital Twins instance");
        }
    });

    //Wait the connection to ADT and then start the adapter
    isConnectedToADT.future().onComplete(ar -> {
        if (ar.succeeded()) {
            //PA connected successfully to the Azure Digital Twins
                instance
```

```
            System.out.println("Physical Asset Adapter connected to
                ADT");
            //Create listener to start shadowing process for data
            eventBus.consumer(
                PhysicalAssetAdapter.PA_SHADOWING_NEW_DATA,
                this::handleMessage);
            //Create listener to start shadowing process for
                relationships
            eventBus.consumer(
                PhysicalAssetAdapter.PA_SHADOWING_RELATIONSHIPS,
                this::handleRelMessage);
        }
    });
}
```

Listato 6.4: Codice principale PhysicalAssetAdapter

Il PhysicalAssetAdapter ha il compito di utilizzare le API di Azure Digital Twins attraverso l'SDK fornito da Microsoft. È necessario innanzitutto connettersi al servizio con le credenziali create nella fase di deploy. Questa azione può richiedere diversi secondi e visto che i verticle di Vert.x sono basati su un event-loop essa non può essere eseguita sul thread principale che altrimenti si bloccherebbe venendo a meno alla regola principale di Vert.x: "non bloccare l'event-loop". Quindi, è necessario eseguirla all'interno di un pool di worker thread attraverso l'utility `vertx.executeBlocking` che esegue in modo asincrono una porzione di codice potenzialmente bloccante. Sfruttando il meccanismo delle *Promise* e dei *Future* è possibile capire, in modo asincrono, il momento in cui la connessione è avvenuta con successo e quindi in cui poter definire i *consumer sull'event bus* per ricevere gli eventi dal DataHub. I dati ricevuti verranno processati dalle rispettive funzioni `handleMessage` ed `handleRelMessage`. La fase di connessione, le operazioni di aggiornamento dei Digital Twins e la gestione delle relazioni avvengono, sempre in modo asincrono, grazie ad un modulo che gestisce la comunicazione con Azure Digital Twins. Tenendo in

considerazione che esso deve essere utilizzato sia dal PA che dall'EWA, visto che entrambi hanno necessità di utilizzare l'SDK, ho preferito creare un'unica classe per gestire tutto ciò, applicando però il principio "ISP" (Interface Segregation Principle). Questo principio stabilisce che le classi non vanno forzate a dipendere da metodi che non servono. Infatti, ove possibile, è opportuno costruire delle interfacce specifiche per le richieste di ogni tipologia di cliente, con solo i metodi usati. Questo significa che, essendo PA e EWA due clienti diversi della classe che gestisce la comunicazione con Azure Digital Twins e avendo due tipologie di permessi diverse (PA può accedere solo agli aggiornamenti di stato e alle relazioni, mentre EWA solo alle operazioni che riguardano i servizi offerti tramite API) ho applicato questo principio in modo tale che ogni attore abbia a disposizione solamente i metodi di cui effettivamente ha bisogno. La struttura è quindi quella descritta in Figura 6.3.

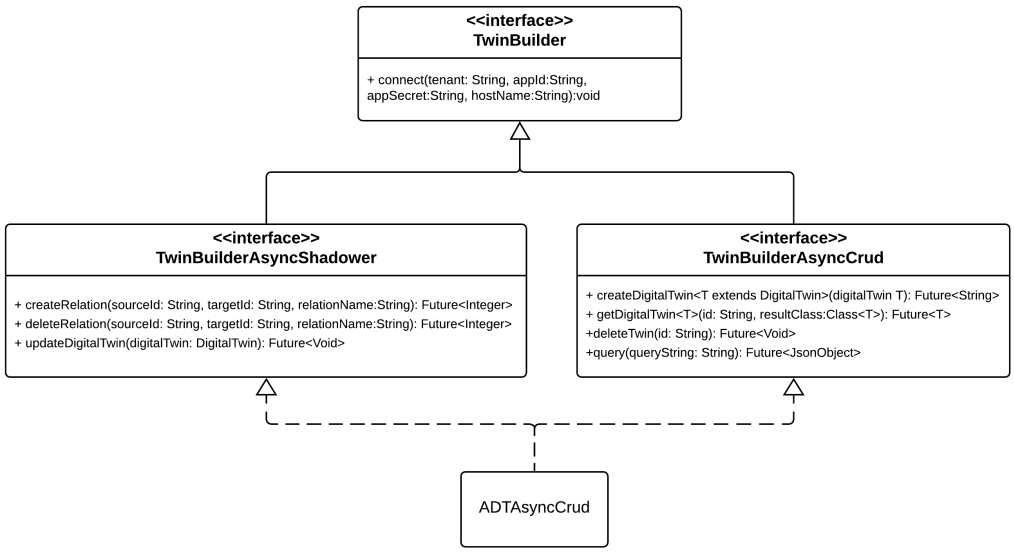

Figura 6.3: Schema UML dell'approccio ISP per il Twin Builder

Viene definita un'interfaccia che modella il comportamento base (`TwinBuilder`) la quale viene poi estesa da due interfacce ognuna delle quali definisce una tipologia di cliente: `TwinBuilderAsyncShadower` e `TwinBuilderAsyncCrud`. Queste interfacce vengono poi implementate da `ADTAsyncCrud`.

Inoltre, si nota come il PhysicalAssetAdapeter, così poi come anche l'EWA, applichi anche il principio "DIP" (Dependency Inversion Principle) in modo da non dipendere dall'aspetto implementativo e quindi dall'utilizzo di Azure Digital Twins come Twin Builder, ma dipendere solamente dalle funzionalità offerte. Perciò, nella creazione dell'oggetto, viene specificato *TwinBuilderAsyncShadower* come tipo in modo da dipendere univocamente dall'interfaccia (che non presenta alcun elemento proveniente dalla libreria di Microsoft) consentendo, allo stesso tempo, di completare l'applicazione del principio ISP definendo l'attore *Shadower* per l'utilizzo della classe.
Un esempio di richiesta tramite l'SDK di Azure Digital Twins, effettuata internamente alla classe *ADTAsyncCrud*, è la seguente:

```
private DigitalTwinsAsyncClient digitalTwinClient;

@Override
public final <T> Future<T> getDigitalTwin(final String id, final
    Class<T> resultClass) {
    final Promise<T> promise = Promise.promise();
    try {
        this.digitalTwinClient.getDigitalTwin(id, resultClass)
        .doOnSuccess(res -> {
            promise.complete(res);
        }).doOnError((e) -> {
            promise.fail(e);
        }).subscribe();
    } catch (Exception e) {
        System.out.println("Error ADTCrud: " + e);
        promise.fail(e);
    }
    return promise.future();
}

@Override
public final Future<Void> deleteTwin(final String id) {
```

```
    final Promise<Void> promise = Promise.promise();
    try {
        CompositeFuture.all(this.deleteIncomingRelationShips(id),
            this.deleteOutgoingRelationShips(id))
        .onSuccess(ar -> {
            System.out.println("Relationships deleted");
            this.digitalTwinClient.deleteDigitalTwin(id)
            .doOnSuccess(res -> {
                promise.complete();
            }).doOnError(err -> {
                System.out.println("Error ADTCrud: " + err);
                promise.fail(err);
            }).subscribe();
        }).onFailure(e -> {
            promise.fail(e);
        });
    } catch (Exception e) {
        System.out.println("Error ADTCrud: " + e);
        promise.fail(e);
    }
    return promise.future();
}

private Future<Void> deleteIncomingRelationShips(final String id) {
    final Promise<Void> promise = Promise.promise();
    final List<Mono<Void>> delTasks = new ArrayList<>();
    try {
        this.digitalTwinClient.listIncomingRelationships(id)
        .doOnNext(incomingRel -> {
            //Add task to delete this relationship
            delTasks.add( this.digitalTwinClient.deleteRelationship(
                incomingRel.getSourceId(),
                incomingRel.getRelationshipId()
                ).subscribeOn(Schedulers.boundedElastic()));
```

```
        })
        .doAfterTerminate(() -> {
            //When we start all the delete tasks, subscribe to them
                and when all completed we can complete the promise.
            if (!delTasks.isEmpty()) {
                System.out.println("Deleted Incoming");
                Mono.when(delTasks)
                    .doOnSuccess(res -> promise.complete())
                    .doOnError(err -> {
                        System.out.println(err);
                        promise.fail(err);
                    }).subscribe();
            } else {
                System.out.println("Incoming rel empty");
                promise.complete();
            }
        }).subscribe();
    } catch (Exception e) {
        System.out.println(e);
        promise.fail(e);
    }
    return promise.future();
}
```

Listato 6.5: Codice principale ADTAsyncCrud

Si può notare come sia tutto completamente asincrono e reattivo, in linea con il pattern di Vert.x. Il primo metodo consente di ottenere lo stato completo di un Digital Twin. Vengono usati i generici per far scegliere all'utilizzatore la classe con cui vuole serializzare il risultato. È possibile utilizzare le classi che descrivono i modelli di Digital Twin, come descritto tra poco, in quanto sono in grado di serializzare e de-serializzare il contenuto. Oppure, può essere richiesto semplicemente come stringa e allora in quel caso verrà restituito in formato JSON. Il secondo metodo consente di eliminare un Digital Twin. Prima di eli-

minare un Digital Twin, Azure Digital Twins richiede che tutte le relazioni in ingresso e in uscita vengano eliminate. Per questo motivo, prima dell'eliminazione, vengono lanciati altri due jobs asincroni per l'eliminazione delle relazioni (ne viene mostrato solo uno in quanto l'altro è pressoché identico). Dopodiché, il metodo, sfruttando i `CompositeFuture`, registra un handler per il successo (e anche per il caso di fallimento) al fine di procedere poi all'eliminazione vera e propria del Digital Twin (sempre in modo asincrono).

Inoltre, sia per quanto riguarda il processo di aggiornamento di un Digital Twin sia per le operazioni che deve svolgere l'EWA è necessario adattare i dati al modello e quindi provvedere un convertitore che, a partire dai dati e dal tipo di modello, restituisca un oggetto che rappresenti il Digital Twin. Modellare ogni tipo di Digital Twin in una classe offre notevoli vantaggi:

- Serializzazione/De-serializzazione: è molto più semplice eseguire queste operazioni richieste dall'SDK di Microsoft ed eventualmente necessarie in futuro.

- Possibilità di definire il comportamento: utilizzando una classe si ha il naturale vantaggio di poter definire i comportamenti e quindi è da prendere in considerazione per sviluppi futuri.

Al fine di applicare anche qui il principio "DIP" viene definita un'interfaccia (presente anche nello schema e nel codice mostrato precedentemente) di nome "DigitalTwin". Essa viene implementata da una classe astratta che definisce i metodi comuni a tutti i Digital Twins. In questo modo, per definire una classe che rappresenti un nuovo tipo di Digital Twin è sufficiente estendere la classe astratta e definire le parti di codice necessarie ed, eventualmente, quelle aggiuntive.
Un possibile convertitore, che a partire dal tipo del Digital Twin restituisca un oggetto di tale classe, potrebbe essere il seguente:

```
public Optional<DigitalTwin> fromType(final String type) {
    Optional<DigitalTwin> res = Optional.empty();
    switch (type) {
```

```
        case "patient":
            res = Optional.of(new Patient());
        break;
        case "ambulance":
            res = Optional.of(new Ambulance());
        break;
        case "mission":
            res = Optional.of(new Mission());
        break;
        default:
            throw new IllegalArgumentException();
    }

    return res;
}
```

Listato 6.6: Possibile convertitore da tipo di Digital Twin ad oggetto della classe corrispondente

Riassumendo, il Physical Asset Adapter si collega al Twin Builder, crea gli handler per gli eventi provenienti dal DataHub, elabora i dati convertendoli nei modelli necessari e permette quindi il completamento del processo di shadowing.

L'External World Adapter invece, è formato da tre verticle: *ApiComponent*, *EventObserverClient* e *CoreEventDispatcher*.
L'ApiComponent è il verticle che si occupa della gestione delle API del nodo con la quale i vari agenti interagiscono. Sfruttando i Router e l'HTTPServer offerto da Vert.x è stato possibile creare agilmente un servizio REST. Una porzione di codice della classe è la seguente (non inserisco il codice completo per motivi di spazio):

```
private Map<String, Boolean> validityMap = new HashMap<>();
private TwinBuilderAsyncCrud twinBuilderCrud;
```

```
@Override
public final void start() {
    final EventBus eventBus = vertx.eventBus();
    Router router = Router.router(vertx);
    router.route().handler(BodyHandler.create());
    //Get Digital Twin API
    router.get("/api/twins/:dtId").handler( this::getDigitalTwin);
    //Create Digital Twin API
    router.post("/api/twins/:type/").handler(
        this::createDigitalTwin);
    //Delete Digital Twin API
    router.delete("/api/twins/:dtId").handler(
        this::deleteDigitalTwin);
    //Query Digital Twin Graph API
    router.post("/api/twins/sparql").handler( this::queryGraph);
    //Get Digital Twin Events API
    router.get("/api/twins/:dtId/events").handler(
        this::getDigitalTwinEvents);

    //Set-up twinBuilderCrud and connect
    this.twinBuilderCrud = new ADTAsyncCrud();
    //Wait the connection to ADT and then start the API service
    this.connectToTwinBuilder().onComplete(ar -> {
        if (ar.succeeded()) {
            //EWA connected successfully to the Azure Digital Twins
                instance
            System.out.println("External World Adapter connected to
                ADT");
            //Start the http server
            vertx.createHttpServer()
                .requestHandler(router)
                .listen(ApiComponent.PORT);
            System.out.println("Http Server on");
        }
```

```
    });

    eventBus.consumer(ApiComponent.EVENT_BUS_DT_VALIDITY, message ->
        {
        final JsonObject jsonObj = new
            JsonObject(message.body().toString());
        final String id =
            jsonObj.getString(CoreEventDispatcher.ID_LABEL);
        final boolean validity =
            jsonObj.getBoolean(CoreEventDispatcher.VALID_LABEL);
        this.validityMap.put(id, validity);
        System.out.println(id + " - " + (validity ? "VALID" : "NOT
            VALID"));
    });
}

private void getDigitalTwinEvents(final RoutingContext
    routingContext) {
    System.out.println("event request");
    HttpServerResponse response = routingContext.response();
    final EventBus eventBus = vertx.eventBus();
    final String dtId = routingContext.pathParam("dtId");
    //Create the web socket from the original request
    routingContext.request().toWebSocket()
    .onSuccess(ws -> {
        System.out.println("WebSocket created for Digital Twin: " +
            dtId);
        //Consume event for this Digital Twin id.
        eventBus.consumer(EventObserverClient.EVENT_OBSERVER_PREFIX +
            dtId, data -> {
            //Write data to web socket
            ws.writeTextMessage(data.body().toString());
        });
    }).onFailure((e) -> {
```

```
        response.setStatusCode(HttpCode.BAD_REQUEST.getCode()).end();
    });
}

private void getDigitalTwin(final RoutingContext routingContext) {
    System.out.println("get request");
    HttpServerResponse response = routingContext.response();
    final String dtId = routingContext.pathParam("dtId");

    this.twinBuilderCrud.getDigitalTwin(dtId, String.class)
    .onSuccess(stringResult -> {
        final JsonObject additionalProps = new JsonObject();
        additionalProps.put("inSynch",
            this.validityMap.getOrDefault(dtId, true));
        //Convert to RDF
        final String rdfResult =
            JSONToRDF.jsonGetDTResponseToRDF(stringResult,
            additionalProps);
        response.putHeader("content-type", "text/plain")
        .setStatusCode(HttpCode.OK.getCode()).end(rdfResult);
    }).onFailure(err -> {
        //Understand failure
        if (this.isDigitalTwinNotFoundException(err)) {
            final JsonObject returnedError = new JsonObject();
            returnedError.put("error", "Digital Twin Id not found.");
            response.putHeader("content-type", "application/json")
                    .setStatusCode(HttpCode.NOT_FOUND.getCode())
                    .end(returnedError.encode());
        } else {
            response.setStatusCode(HttpCode.SERVER_ERROR.getCode())
            .end();
        }
    });
}
```

Listato 6.7: Porzione di codice del verticle ApiComponent

È possibile notare l'utilizzo del tipo *TwinBuilderAsyncCrud* (sempre al fine di applicare il principio ISP) e la presenza di una mappa per memorizzare la validità dei Digital Twins. Questa mappa è solo a fini dimostrativi, può essere sostituita agilmente da un qualsiasi tipo di storage. L'ApiComponent inizialmente imposta le varie route dei servizi offerti, dopodiché, dovendo dialogare con il Twin Builder, apre una connessione con quest'ultimo (utilizzando le credenziali create per l'accesso ad Azure Digital Twins da parte dell'EWA). Una volta connesso, crea il server HTTP per poter gestire le richieste degli agenti esterni.
Nella porzione 6.7 è possibile vedere il codice che soddisfa due delle richieste possibili. Il primo handler definito è per la richiesta di osservazione di un Digital Twin. Come si può vedere, una volta ricevuta la richiesta la converte in una Web Socket (se non vi sono problemi) e crea un consumer sull'event bus al fine di mettersi in ascolto degli eventi riguardanti il Digital Twin d'interesse. Infatti, l'EventObserverClient invierà tutti gli eventi raccolti dal servizio SignalR sull'event-bus ad un indirizzo che dipende dall'id del Digital Twin coinvolto. In questo modo, ogni volta che arriva un messaggio questo verrà ritrasmesso attraverso la socket all'agente. Il secondo handler invece, è per la richiesta dello stato completo di un Digital Twin. Come da requisito, è necessario ritornare il risultato in RDF, perciò è necessario utilizzare un convertitore che trasformi i dati, restituiti in JSON dal metodo definito in TwinBuilderAsyncCrud, in triple RDF. A tal fine, è stato sviluppato un convertitore che, sfruttando Apache Jena, preso in input lo stato in JSON, restituisce lo stesso stato in RDF.
Per fornire il servizio di query, come anticipato nei requisiti e nella proposta di architettura, è necessario convertire le query SPARQL fornite dall'agente in query espresse nell'SQL di Azure Digital Twins. Inoltre, è anche necessario un ulteriore convertitore che a partire dal risultato della query in JSON ottenga gli stessi dati in formato SPARQL-JSON rispettando la recommendation [20]. La creazione di un convertitore da SPARQL a SQL (di Azure Digital Twins,

un SQL ad-hoc) è un lavoro molto oneroso che richiede la comprensione della query. Il formato SQL di Azure Digital Twins non è standard ed ha alcune regole discordi dall'SQL tradizionale. Tuttavia, ispirandomi per le fasi iniziali a [4] ho creato una prima versione di convertitore che permette l'esecuzione di query di base, senza le funzionalità "OPTIONAL" o "'FILTER", anche se, in teoria potrebbero essere aggiunte.
I passi da seguire per implementare l'algoritmo sviluppato all'intero del progetto sono i seguenti (non riporto il codice intero per motivi di dimensioni):

1. Dividere la query nei suoi componenti principali al fine di poter eseguire le elaborazioni necessarie. Separare quindi la parte di "SELECT" e quella di "WHERE" e dalla prima ottenere tutte le *question word* richieste nel risultato.

2. A partire dal blocco WHERE, ottenere tutti gli statement presenti.

3. Rappresentare ogni statement attraverso una modellazione ad oggetti interpretando ogni sua parte con i rispettivi ruoli quindi, Soggetto, Predicato e Oggetto indicando ogni qual volta uno di questi elementi si presenta come question word. Inoltre, occorre categorizzare i predicati in modo da stabilire se rimandano ad un *Object property* (quindi ad una relazione) oppure ad un *DataType property* (quindi ad una proprietà).

4. Da qui iniziano ad essere applicate le prime regole statiche per la conversione. È innanzitutto necessario individuare tutti gli elementi che svolgono il ruolo di soggetto. L'unico soggetto che ricopre solamente tale ruolo all'interno del graph pattern (e quindi è sempre soggetto nelle triple del pattern) è il modello di Digital Twin di riferimento nella query SQL di Azure Digital Twins (*FROM DIGITALTWINS T*). Gli altri, quindi gli elementi che fanno sia da soggetto che da oggetto, riguardano le relazioni tra Digital Twins. Infatti, se un elemento appare sia come oggetto che come soggetto, ed è, come nella maggior parte dei casi, una question word, significa che è stato prima un Object property e poi anche un soggetto, quindi si è verificano nel graph pattern un legame con

il soggetto del predicato in cui compare come oggetto. Un esempio per comprendere meglio il concetto è il seguente:

```
SELECT ?movie ?director
WHERE {
    :James :playsIn ?movie .
    ?movie :directedBy ?director .
}
```

Listato 6.8: Esempio relazione

In questo caso è possibile notare la question word "?movie" ricoprire ruolo sia di oggetto (nel primo statement del graph pattern) sia quello di soggetto (nel secondo statement), perciò, in termini di query Azure Digital Twins, quella è una relazione con il soggetto dello statement in cui compare come oggetto ("*:James :playsIn ?movie*", soggetto "James" e nome della relazione "playsIn") che deve essere tradotta in SQL di Azure Digital Twins attraverso un "JOIN".

5. Nel caso in cui una question word appaia solo come soggetto e sia anche tra le variabili nella SELECT, perciò da ritornare nel risultato, questa verrà aggiunta alla SELECT della nuova query ottenendo l'id (*Alias.$dtId AS "question word"*) del Digital Twin in questione.

6. Per ogni question word che compare solo come oggetto è necessario individuare il suo soggetto (e quindi il suo alias) e se questa compare anche nella SELECT allora deve essere aggiunta anche nella SELECT SQL come:

   Alias-soggetto.nome-predicato AS "question word oggetto"

7. Si analizza ogni statement che compare nella clausola WHERE e si cercano tutti i valori impostati all'interno del pattern (ad esempio *mfg:Product1* oppure *"New York"* con eventuale estensione *xsd:string*). In questi casi, se l'elemento è soggetto si aggiunge un controllo all'interno della clausola WHERE SQL:

Alias-soggetto.$dtId = {elemento}

Altrimenti, nel caso in cui sia un oggetto di tipo “DataType property” si aggiunge alla clausola WHERE SQL:

Alias-soggetto.nome-predicato = {elemento}

Infine, nel caso in cui sia un oggetto di tipo “Object property”, quindi una relazione, è necessario aggiungere un JOIN con cui esprimere la relazione a livello di Digital Twins.

Un esempio, che mette in risalto le parti principali della query e ne offre la conversione elaborata dall’algoritmo, è in Figura 6.4. Qui possiamo notare

```
SELECT ?mission ?patient ?heart
WHERE {
   ?mission ex:vehicle :Ambulance1 .
   ?mission ex:patient ?patient .
   ?patient ex:heartRate ?heart
}

SELECT T.$dtId AS mission,  patient.$dtId AS patient,  patient.heartRate AS heart
FROM DIGITALTWINS T
JOIN patient RELATED T.patient
JOIN vehicle RELATED T.vehicle
WHERE vehicle.$dtId = 'Ambulance1'
```

Figura 6.4: Esempio di conversione

una query in cui non viene specificato il soggetto, ma solamente l’oggetto di una relazione e quindi tramite quello si vogliono filtrare i soggetti per poter accedere ad un’altra relazione del soggetto ed ottenere dei dati.
Lo schema del grafo di conoscenza per capire il contesto è il seguente (Figura 6.5):
Quindi, come si può vedere vi è un salto tra due relazioni all’interno della query e il convertitore è stato in grado di comprenderli e trasformarli adeguatamente. Era necessario capire che i predicati *ex:vehicle* e *ex:patient* facessero

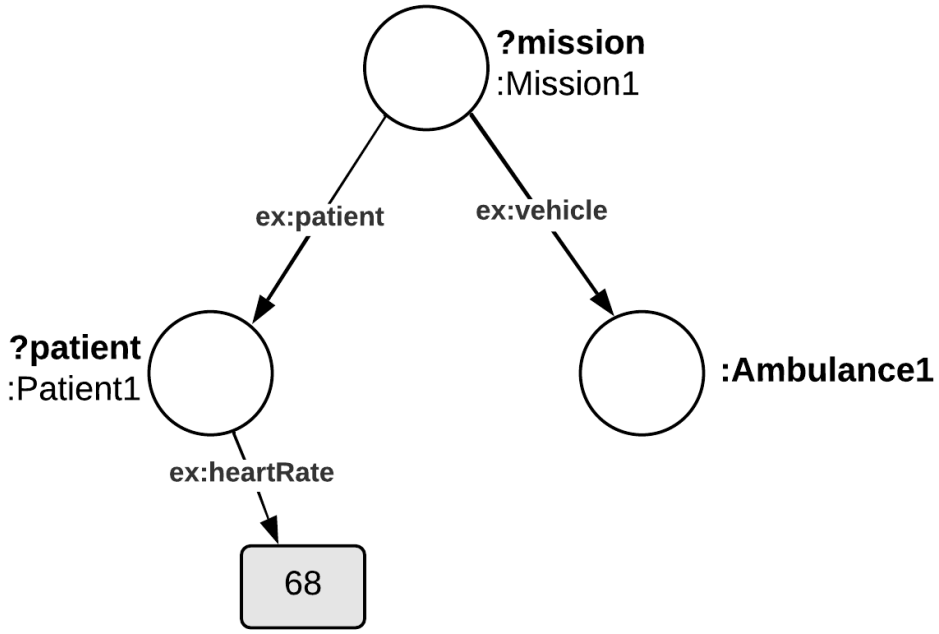

Figura 6.5: Grafo descrittivo per la query di esempio

riferimento ad una "Object property". Questo è stato possibile in quanto il primo riporta un oggetto senza alcun tipo di formato indicato e soprattutto come URI nel formato CURIE. Per il secondo invece, il convertitore ha visto che la question word era presente sia nel ruolo di oggetto e sia in quello di soggetto, quindi attraverso le regole statiche definite è riuscito a dare forma alla relazione tra Digital Twins.

Un ulteriore esempio, più semplice, che prevede l'ottenimento di alcune informazioni di stato da parte di un soggetto specificato attraverso un URI è descritto in Figura 6.6.

Questi sono solo due esempi dell'elaborazione del convertitore. Con questo si vuole dimostrare l'applicabilità del concetto definito in modo astratto all'interno dei requisiti e nella proposta architetturale. Ovviamente, l'algoritmo può essere migliorato e sicuramente ci saranno sviluppi futuri al fine di poter utiliz-

```
SELECT ?location ?name
WHERE {
   mfg:Product1 mfg:productManifactureLocation ?location .
   mfg:Product1 rdf:label ?name .
}

SELECT T.label AS name,  T.productManifactureLocation AS location
FROM DIGITALTWINS T
WHERE T.$dtId = 'Product1'
```

Figura 6.6: Ulteriore esempio di conversione

zare sempre più funzionalità di SPARQL all'interno delle query. OPTIONAL e FILTER per ora non sono stati implementati solo per questioni di tempo, anche se, a livello di pseudocodice erano già stati previsti e la loro individuazione doveva avvenire subito dopo l'ultimo step definito precedentemente.

Oltre a ciò, è presente anche un convertitore che a partire dai dati in JSON, restituiti da Azure Digital Twins, ottiene il risultato in SPARQL-JSON. Ad esempio, rispetto alla query illustrata in Figura 6.4, è possibile ottenere il seguente risultato:

```
{
    "head": {
        "vars": [
            "mission",
            "patient",
            "heart"
        ]
    },
    "results": {
        "bindings": [
            {
                "mission": {
                    "value": "Mission1"
```

```
            },
            "patient": {
                "value": "Patient1"
            },
            "heart": {
                "value": 68
            }
        }
    ]
  }
}
```

Listato 6.9: Conversione JSON in SPARQL-JSON

In questo modo è possibile nascondere all'agente la tecnologia utilizzata internamente.

Come detto precedentemente, l'EventObserverClient deve connettersi al servizio SignalR per ricevere gli eventi in real-time dei Digital Twins e metterli a disposizione delle web sockets consentendo a quest'ultime di notificare gli agenti interessati. La parte principale di questo componente è la seguente:

```
final EventBus eventBus = vertx.eventBus();
final HubConnection hubConnection = HubConnectionBuilder
        .create(EventObserverClient.SIGNALR_NEGOTIATE_FUNCTION_URL)
        .build();

//After building and before start the connection, define the hub
    listeners
hubConnection.on("newMessage", (message) -> {
    System.out.println("New Message from SignalR: " + message);
    this.getId(message).forEach((id) -> {
        eventBus.publish(EventObserverClient.EVENT_OBSERVER_PREFIX +
            id, this.getEventObjFromAdtEvent( new
            JsonObject(message)).encode());
    });
```

```
}, String.class);

hubConnection.start().doOnComplete(() -> {
    System.out.println("Connected to SignalR!");
}).doOnError((e) -> {
    System.out.println("Error SignalR: " + e);
}).subscribe();
```

Listato 6.10: Porzione di codice schema di EventObserverClient

Si nota come la prima azione che porta a termine sia la creazione della connessione al servizio SignalR. In essa è necessario specificare l'endpoint per il processo di negoziazione che corrisponde all'URL della funzione con HTTP-Trigger definita in Azure Function (URL ottenibile da Azure Portal). Dopodiché, provvede ad impostare l'hanlder per la ricezione dei nuovi messaggi. Ogni messaggio ricevuto viene analizzato al fine di ottenere tutti gli id dei Digital Twins coinvolti. Per ognuno di essi viene inviato un messaggio sull'event bus ad un indirizzo che comprende l'id in questione. In questo modo, il processo che gestisce la web socket per un'osservazione in corso, mettendosi in ascolto su un indirizzo specifico (quello dell'id interessato), riceverà solo gli eventi di rilievo e quindi di interesse per l'agente collegato.

Infine, è presente un verticle che si occupa di ricevere tutti gli eventi d'interesse "da e per" il Core al fine di mantenere sincronizzate ed aggiornate le due parti. Questo è il CoreEventDispatcher il quale riceve eventi dai componenti del nodo attraverso l'event bus di Vert.x e li spedisce direttamente al Core tramite il canale bidirezionale che viene costruito non appena il componente si avvia. Inoltre, il CoreEventDispatcher riceverà anche gli eventi provenienti dal Core sulla validità dei singoli Digital Twin. Il collegamento bidirezionale viene creato sfruttando le `NetSocket` offerte da Vert.x, le quali consentono di creare velocemente delle socket TCP tra due endpoint.

6.2.3 Implementazione Core

Anche il Core è stato sviluppato in Java con il supporto del toolkit Vert.x. I verticles rispecchiano quelli che sono i componenti presenti nella proposta di architettura. Una descrizione schematica si trova in Figura 6.7. Ora procederò alla descrizione delle parti principali mostrando, quando necessario, le parti di rilievo del codice sviluppato.

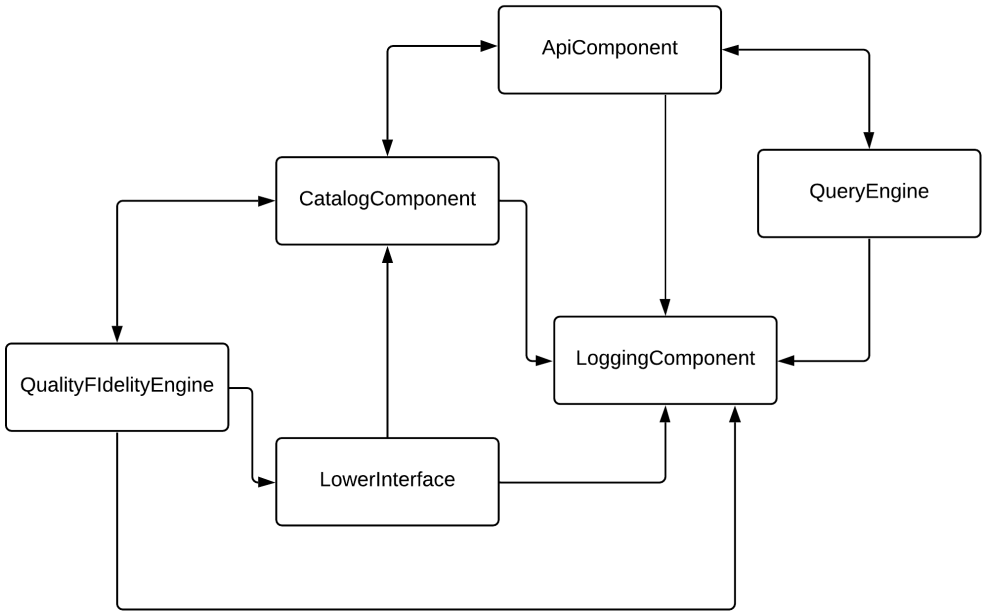

Figura 6.7: Schema dei verticles all'interno del Core

La LowerInterface è responsabile della gestione delle connessioni con i nodi presenti nell'ecosistema e quindi della ricezione e dell'invio dei vari eventi che permettono all'architettura di rimanere consistente e sincronizzata. Essa viene sviluppata attraverso un verticle Vert.x. Si dovrà occupare di rimanere in ascolto per la creazione di nuove socket di comunicazione con i nodi, della ricezione e dello smistamento degli eventi in arrivo e dell'invio di dati alle rispettive organizzazioni. Le socket vengono create, come nel CoreEventDispatcher del Nodo, utilizzando `NetSocket` offerto da Vert.x. e le comunicazioni con gli altri componenti del Core avvengono grazie all'event bus.

```
private final Map<URI, NetSocket> mapping = new HashMap<>();

@Override
public final void start() {
    this.eventBus = vertx.eventBus();
    final NetServer server = vertx.createNetServer();
    System.out.println("Creating server socket...");
    server.connectHandler(socket -> {
        System.out.println("New connection from WoDT NODE");
        socket.handler(buffer -> {
            this.handleNewEvent(buffer.toJsonObject(), socket);
        });
    }).listen(LowerInterface.TCP_SOCKET_PORT, "localhost", res -> {
        if (res.succeeded()) {
            System.out.println("WoDT Core Lower Interface socket
                created");
        } else {
            System.out.println("Error: " + res.cause().getMessage());
            System.exit(0);
        }
    });

    this.eventBus.consumer(LowerInterface.SIGNAL_TO_NODE, msg -> {
        final JsonObject jsonObj = new
            JsonObject(msg.body().toString());
        final URI uri = URI.create(jsonObj.getString(
            LowerInterface.ORG_LABEL));
        final String msgToSend = jsonObj.getString(
            LowerInterface.MSG_LABEL);
        if (this.mapping.containsKey(uri)) {
            this.mapping.get(uri).write(msgToSend);
        }
    });
```

```
}
```

Listato 6.11: Codice principale LowerInterface

In questa piccola porzione di codice si nota come all'interno del verticle LowerInterface venga memorizzato il mapping tra l'URI dell'organizzazione e la socket associata. In questo modo, nel caso in cui qualsiasi componente del Core necessiti di inviare dei messaggi ad un'organizzazione, come nel caso degli eventi riguardanti lo stato di validità di un Digital Twin, può semplicemente specificare l'URI e la LowerInterface sarà in grado di individuare il corretto canale di comunicazione in cui inviarli. Dopodiché, non appena il verticle viene avviato, occorre creare la socket TCP di benvenuto per tutte le connessioni in ingresso, cioè per tutti i nodi che desiderano connettersi. Per ogni socket, si stabilisce un handler per i messaggi ricevuti. Questo handler andrà a gestire e a smistare tutti gli eventi in ingresso e memorizzerà il mapping tra URI e Socket. Inoltre, attraverso l'event bus, consentirà la comunicazione asincrona con gli altri componenti del Core. Gli altri componenti, per poter comunicare con i nodi, effettuano una richiesta di invio dati al verticle LowerInterface pubblicando un evento sull'event bus. Perciò, nello start, il verticle LowerInterface creerà anche un handler per tutti gli eventi che arrivano sull'indirizzo dell'event bus opportuno e consentirà a tutti i componenti di dialogare con i nodi fornendo semplicemente l'URI.

Il Catalogo si occupa della gestione dei Digital Twins presenti in tutto l'ecosistema. Come anticipato, i Digital Twins verranno memorizzati in triple RDF a formare un grafo di conoscenza. Per poter gestire il grafo opportunamente, il Catalogo dovrà rispondere agli eventi raccolti dalla Lower Interface provenienti dalle organizzazioni e provvedere anche un servizio che consenta di eseguire operazioni di CRUD sul grafo stesso. Una porzione di codice del Catalogo è la seguente:

```
private EventBus eventBus;
private RdfCrud rdfCrud;
```

```
@Override
public final void start() {
    this.eventBus = vertx.eventBus();
    this.rdfCrud = new RdfCrudImpl();

    //New Digital Twin event consumer
    this.eventBus.consumer(CatalogComponent.CREATE_DIGITAL_TWIN,
        message -> {
        System.out.println("CATALOG COMPONENT - Processing" +
            message.body().toString());
        final JsonObject jsonObj = new
            JsonObject(message.body().toString());
        final List<String> digitalTwinClasses = new ArrayList<>();
        jsonObj.getJsonArray(LowerInterface.CLASSES_LABEL)
            .spliterator().forEachRemaining((cls) -> {
            digitalTwinClasses.add(cls.toString());
        });
        final CoreDigitalTwinBuilder builder = new
            CoreDigitalTwinBuilder();
        builder.setId(jsonObj.getString(LowerInterface.ID_LABEL))
                .setBaseUri(URI.create(jsonObj.getString(
                    LowerInterface.ORG_LABEL)))
                .setClasses(digitalTwinClasses)
                .setLastUpdate(LocalDateTime.now())
                .setFidelityInterval(jsonObj.getInteger(
                    LowerInterface.FIDELITY_LABEL));

        final CoreDigitalTwin digitalTwinToCreate = builder.build();
        this.rdfCrud.save(digitalTwinToCreate);
    });

    //Delete Digital Twin event consumer
    this.eventBus.consumer(CatalogComponent.DELETE_DIGITAL_TWIN,
        message -> {
```

```
        System.out.println("CATALOG COMPONENT - Processing" +
            message.body().toString());
        final JsonObject jsonObj = new
            JsonObject(message.body().toString());

        final CoreDigitalTwinBuilder builder = new
            CoreDigitalTwinBuilder();
        builder.setId(jsonObj.getString(LowerInterface.ID_LABEL))
                .setBaseUri(URI.create(jsonObj.getString(
                    LowerInterface.ORG_LABEL)));
        final CoreDigitalTwin digitalTwinToRemove = builder.build();
        this.rdfCrud.delete(digitalTwinToRemove.getUri());
    });

    //Updated Digital Twin event consumer
    this.eventBus.consumer(CatalogComponent.UPDATED_DIGITAL_TWIN,
        message -> {
        System.out.println("CATALOG COMPONENT - Processing" +
            message.body().toString());
        final JsonObject jsonObj = new
            JsonObject(message.body().toString());
        final CoreDigitalTwinBuilder builder = new
            CoreDigitalTwinBuilder();
        builder.setId(jsonObj.getString(LowerInterface.ID_LABEL))
                .setBaseUri(URI.create(jsonObj.getString(
                    LowerInterface.ORG_LABEL)));
        final CoreDigitalTwin digitalTwinToRemove = builder.build();
        this.rdfCrud.updateLastUpdate(digitalTwinToRemove.getUri(),
            LocalDateTime.now());
    });
}
```

Listato 6.12: Porzione del codice del Catalogo

Si può notare come esso rimanga in ascolto sull'event bus per gli eventi provenienti dalla Lower Interface e a seconda dell'evento agisca di conseguenza. Qui vengono mostrati solo gli eventi di creazione, cancellazione e update, ma sono presenti anche quelli per la gestione delle relazioni. Per eseguire le operazioni sul grafo di conoscenza utilizza una classe esterna, separando la logica di gestione. Questa classe è `RdfCrudImpl` e si può osservare una porzione di codice in 6.13:

```
private Dataset dataset;

public RdfCrudImpl() {
    this.dataset =
        TDBFactory.createDataset(RdfCrudImpl.RDF_STORE_PATH);
}

@Override
public final boolean save(final CoreDigitalTwin digitalTwin) {
    this.dataset.begin(ReadWrite.WRITE);
    try {
        final Model model = this.dataset.getDefaultModel();
        //Create the resource for the new Digital Twin
        final Resource newDigitalTwinRes =
            model.createResource(digitalTwin.getUri());
        //Create the properties
        final Property idProp =
            model.createProperty(RdfCrudImpl.PROPERTY_PREFIX + "id");
        final Property baseProp =
            model.createProperty(RdfCrudImpl.PROPERTY_PREFIX +
            "base");
        final Property classProp =
            model.createProperty(RdfCrudImpl.PROPERTY_PREFIX +
            "class");
        final Property validProp =
            model.createProperty(RdfCrudImpl.PROPERTY_PREFIX +
```

```
            "valid");
        final Property fidelityProp =
            model.createProperty(RdfCrudImpl.PROPERTY_PREFIX +
            "fidelity");
        final Property lastUpdateProp =
            model.createProperty(RdfCrudImpl.PROPERTY_PREFIX +
            "lastUpdate");

        //Save the values
        newDigitalTwinRes.addProperty(idProp, digitalTwin.getId())
            .addProperty(baseProp,
                digitalTwin.getBaseUri().toString())
            .addLiteral(validProp, digitalTwin.isValid())
            .addLiteral(fidelityProp, digitalTwin.getFidelity())
            .addLiteral(lastUpdateProp,
                digitalTwin.getLastUpdate().toString());

        //Iterate classes of Digital Twin to save them.
        digitalTwin.getClasses().forEach((singleClass) -> {
            newDigitalTwinRes.addProperty(classProp, singleClass);
        });

        //Commit the transaction
        this.dataset.commit();
    } catch (Exception e) {
        this.dataset.abort();
        return false;
    } finally {
        //Close the transaction
        this.dataset.end();
    }
    return true;
}
```

```
@Override
public final boolean delete(final String uri) {
    this.dataset.begin(ReadWrite.WRITE);
    try {
        final Model model = this.dataset.getDefaultModel();
        final Resource res = model.getResource(uri);
        System.out.println(res.getURI());
        //Remove all statements where res is subject
        model.removeAll(res, null, null);
        //Remove all statements where res is object (relationships)
        model.removeAll(null, null, res);

        //Commit the transaction
        this.dataset.commit();
    } catch (Exception e) {
        this.dataset.abort();
        System.out.println(e);
        return false;
    } finally {
        //Close the transaction
        this.dataset.end();
    }
    return true;
}
```

Listato 6.13: Porzione del codice del RdfCrudImpl

Per fornire le sue funzionalità utilizza la libreria *Apache Jena* che fornisce anche un *RDFStore* chiamato *TDB*. Infatti, si nota come nel costruttore venga subito creata la connessione al *Dataset* contenente il grafo di conoscenza. Qui vengono mostrate due semplici operazioni: save e delete. Nella prima, si può notare come a partire da un oggetto `CoreDigitalTwin` (descritto successivamente) si apra una transazione in scrittura all'interno dell'RDFStore al fine di poter aggiornare il grafo di conoscenza che in Apache Jena corrisponde al

`Model`. Si procede alla creazione della nuova risorsa, si aggiungono tutte le proprietà necessarie ed infine, si procede al commit della transazione. Per quanto riguarda l'operazione di eliminazione invece, viene ottenuto il modello e la risorsa corrispondente e vengono rimossi tutti gli statement in cui compare come soggetto o come oggetto, sempre all'interno di una transazione in scrittura.

Si può notare nella porzione di codice del Catalogo (6.12) e in 6.13 come alcuni metodi dell'`RdfCrud` prendano in ingresso un oggetto `CoreDigitalTwin`. Ho preferito modellare le informazioni ricevute per un DigitalTwin con una classe, in modo tale che `RdfCrud` fosse indipendente dal formato dei messaggi. Dato l'alto numero di campi, per la creazione di questo oggetto ho utilizzato il pattern Builder. Esso mi ha consentito inoltre, di poter esprimere gli oggetti con solo le proprietà di interesse per il caso d'uso, come si può notare nei vari handler nella porzione di codice 6.12.

Mentre il Core esegue i suoi compiti, vi è un processo che viene eseguito in modo continuo. Questo è il controllo sulla validità dei Digital Twins presenti all'interno dell'ecosistema. Il verticle incaricato di tale controllo è il QualityFidelityEngine e viene eseguito grazie ad una funzionalità offerta da Vert.x chiamata `vertx.setPeriodic` che consente di specificare una porzione di codice da eseguire periodicamente. Il QualityFidelityEngine, eseguendo una query sul grafo di conoscenza, ottiene i Digital Twins e i loro dati. Analizzandoli, verifica se il loro stato di validità è cambiato e, in caso, crea un evento che invia alla Lower Interface attraverso l'event bus al fine di notificare l'organizzazione (quindi il nodo) d'interesse.

Al fine di eseguire le query è necessario creare un engine separato, quindi un verticle (QueryEngine) che gestisca le richieste e le risposte in modo asincrono. Questo perché ogni query può impiegare diversi secondi e l'event-loop di Vert.x non può essere bloccato, perciò è necessario che venga eseguito tutto in modo asincrono delegando il lavoro ad un pool di worker thread. Il Query engine dovrà rimanere in ascolto per tutte le richieste di query provenienti dai componenti interni dopodiché, non appena la query ha terminato l'esecuzione, dovrà restituire il risultato in formato SPARQL-JSON in modo asincrono al mittente della richiesta. Questi scambi di messaggi avvengono sempre utiliz-

zando l'event bus offerto da Vert.x.
Ecco una porzione di codice che consente di eseguire una query sul dataset RDF e di ottenere direttamente il risultato in SPARQL-JSON:

```
this.dataset.begin(ReadWrite.READ);
JsonObject jsonResult = new JsonObject();
try {
    try (QueryExecution qExec = QueryExecutionFactory.create(query,
        this.dataset)) {
        final ResultSet rs = qExec.execSelect();
        final OutputStream outputStream = new ByteArrayOutputStream();
        ResultSetFormatter.outputAsJSON(outputStream, rs);
        jsonResult = new JsonObject(outputStream.toString());
    }
} catch (Exception e) {
    this.dataset.abort();
    System.out.println(e);
    jsonResult.put("error", "invalid query");
} finally {
    this.dataset.end();
}
```

Listato 6.14: Porzione di codice per l'esecuzione di una query

Il Core mette a disposizione, come definito nella proposta architetturale, delle API. Questo servizio è reso disponibile dal verticle ApiComponent. Esso sfrutta i componenti interni e offre i servizi all'esterno in maniera simile a quanto già mostrato per il componente ApiComponent interno al Nodo.

Infine, è presente anche il verticle LoggingComponent che si occupa di tracciare tutti gli eventi interni all'architettura e consente di poter eseguire analisi in caso di problemi al sistema.

Capitolo 7

Valutazione e considerazioni

Dopo aver mostrato l'implementazione delle parti principali, è di fondamentale importanza procedere alla valutazione del sistema e alla dimostrazione dell'applicabilità attraverso un caso di studio che ne metta in risalto i vantaggi e gli svantaggi. Perciò, in questo capitolo verrà descritto il piccolo caso di studio implementato per la verifica del middleware e dell'architettura a supporto e verranno discusse alcune considerazioni su spunti e lavori futuri necessari per incrementare le potenzialità del sistema e per raggiungere gli obiettivi di *Web of Digital Twins*.

7.1 Piccolo caso di studio

Per verificare il funzionamento del sistema sviluppato è stato creato un piccolo caso di studio su cui testarlo. È volutamente piccolo in quanto grazie ad esso sono riuscito a provare le principali funzionalità dell'architettura. Sviluppare un caso di studio più grande e completo non era, in questo momento, la priorità ed è un aspetto che riguarderà prove future.

Il caso di studio è ispirato al caso pratico "Major Trauma Management" descritto in [18] di cui ne verrà presa una piccola parte. Il Major Trauma Management è uno degli scenari di maggiore rilievo interni all'healthcare. Richiede una gestione tempestiva in quanto il trauma grave è il risultato di un

evento (ad esempio un incidente) in grado di causare lesioni tali da delineare un rischio molto elevato per il paziente.

Quello che si vuole modellare qui sono gli attori principali coinvolti nelle fasi iniziali, cioè nella gestione della chiamata di soccorso e nell'individuazione del paziente e dell'ambulanza da coinvolgere nella missione. Infatti, saranno proprio questi tre i Digital Twins coinvolti:

- Missione: non appena arriva la chiamata, l'operatore colleziona le prime informazioni sull'evento accaduto. A partire da queste, crea la *Missione* a cui viene associato un luogo, uno stato, l'ambulanza assegnata e il paziente interessato. Ovviamente, questa è una semplificazione e il modello completo del fenomeno potrebbe richiedere una fase di analisi che costituirebbe una tesi a sé stante.

- Paziente: è il paziente che ha subito il trauma. Se ne rappresentano i dati principali come il battito cardiaco, la pressione del sangue, la temperatura corporea e il codice di gravità.

- Ambulanza: è l'ambulanza incaricata di raggiungere il paziente e portarlo nell'ospedale selezionato. Gli aspetti presi in considerazione nella modellazione sono la posizione corrente in termini di latitudine e longitudine, uno stato che indica quando l'ambulanza è effettivamente occupata e la percentuale di carburante (importante per il raggiungimento del luogo della missione rispetto alla posizione corrente e alla posizione dell'ospedale selezionato).

Con solo questi tre Digital Twins, è possibile testare un gran numero di funzionalità dell'ecosistema: creazione di un'istanza di Digital Twin, creazione di relazioni, aggiornamento tramite adapters (mqtt e http), ottenimento stato completo di un Digital Twin, osservazione di un Digital Twin, esecuzione di query locali all'organizzazione su più relazioni, controllo della fidelity e tanto altro..
I tre Digital Twins sono tutti all'interno della stessa organizzazione e quindi vengono memorizzati all'interno del nodo sviluppato con Azure Digital Twins.

Tenendo in considerazione che l'upload di modelli non è stato considerato all'interno delle funzionalità, in quanto è ancora un argomento di studio che verrà analizzato nelle considerazioni successivamente, ho creato i modelli in DTDL per poterne fare l'upload su Azure Digital Twins. È possibile osservarne la modellazione di seguito:

```
{
    "@id" : "dtmi:com:contoso:patient;1",
    "@type" : "Interface",
    "@context" : "dtmi:dtdl:context;2",
    "displayName" : "Patient",
    "contents" : [
        {
            "@type" : "Property",
            "name" : "heartRate",
            "schema" : "integer"
        },
        {
            "@type" : ["Property", "Pressure"],
            "name" : "bloodPressure",
            "unit" : "millimetresOfMercury",
            "schema" : "double"
        },
        {
            "@type" : ["Property", "Temperature"],
            "name" : "bodyTemperature",
            "unit" : "degreeCelsius",
            "schema" : "double"
        },
        {
            "@type" : "Property",
            "name" : "patientCode",
            "schema" : {
                "@type" : "Enum",
```

```
            "valueSchema" : "integer",
            "enumValues" : [
                {
                    "name" : "red",
                    "displayName" : "Red",
                    "enumValue" : 0
                },
                {
                    "name" : "yellow",
                    "displayName" : "Yellow",
                    "enumValue" : 1
                },
                {
                    "name" : "green",
                    "displayName" : "Green",
                    "enumValue" : 2
                },
                {
                    "name" : "white",
                    "displayName" : "White",
                    "enumValue" : 3
                }
            ]
        }
    }
  ]
}
```

Listato 7.1: Modello DTDL paziente

```
{
    "@id" : "dtmi:com:contoso:ambulance;1",
    "@type" : "Interface",
    "@context" : "dtmi:dtdl:context;2",
```

```
    "displayName" : "Ambulance",
    "contents" : [
        {
            "@type" : "Property",
            "name" : "fuelPercentage",
            "schema" : "integer"
        },
        {
            "@type" : "Property",
            "name" : "busy",
            "schema" : "boolean"
        },
        {
            "@type" : "Property",
            "name" : "latitude",
            "schema" : "double"
        },
        {
            "@type" : "Property",
            "name" : "longitude",
            "schema" : "double"
        }
    ]
}
```

Listato 7.2: Modello DTDL ambulanza

```
{
    "@id" : "dtmi:com:contoso:mission;1",
    "@type" : "Interface",
    "@context" : "dtmi:dtdl:context;2",
    "displayName" : "Mission",
    "contents" : [
        {
```

```
        "@type" : "Property",
        "name" : "place",
        "schema" : "string"
    },
    {
        "@type" : "Property",
        "name" : "status",
        "schema" : {
            "@type" : "Enum",
            "valueSchema" : "integer",
            "enumValues" : [
                {
                    "name" : "started",
                    "displayName" : "Started",
                    "enumValue" : 0
                },
                {
                    "name" : "movingToPatient",
                    "displayName" : "Moving to Patient",
                    "enumValue" : 1
                },
                {
                    "name" : "rescuing",
                    "displayName" : "Rescuing",
                    "enumValue" : 2
                },
                {
                    "name" : "movingToHospital",
                    "displayName" : "Moving to Hospital",
                    "enumValue" : 3
                },
                {
                    "name" : "completed",
                    "displayName" : "Completed",
```

```
                    "enumValue" : 4
                }
            ]
        }
    },
    {
        "@type" : "Relationship",
        "name" : "vehicle",
        "target" : "dtmi:com:contoso:ambulance;1",
        "minMultiplicity": 0,
        "maxMultiplicity" : 1
    },
    {
        "@type" : "Relationship",
        "name" : "patient",
        "target" : "dtmi:com:contoso:patient;1",
        "minMultiplicity": 0,
        "maxMultiplicity" : 1
    }
]
}
```

Listato 7.3: Modello DTDL missione

Una volta effettuato il deploy del sistema e caricati i modelli all'interno del nodo è possibile utilizzare l'ecosistema attraverso le API fornite.
Simulando uno scenario reale, il primo passo è la creazione del Digital Twin della Missione. Questo può essere eseguito richiamando l'API apposita per la creazione di un Digital Twin:

POST http://dominio/api/twins/mission/

Passando all'interno del body i dati necessari, cioè:

```
{
    "DtId" : "Mission1",
```

```
    "fidelity" : 10000
}
```

Listato 7.4: Body della richiesta di creazione missione

In questo modo, viene creato un Digital Twin del tipo "mission" di id *Mission1* e con una fidelity di dieci secondi. Questo significa che se il Digital Twin non viene aggiornato entro dieci secondi (dall'ultimo aggiornamento), passa in uno stato di non validità. Questo è il controllo che esegue continuamente il *Quality and Fidelity engine*.
Dopodiché, la persona che chiama il centro di emergenza specificherà il luogo della missione, quindi verranno eseguite due operazioni:

1. Aggiornamento stato della missione: la missione può essere aggiornata con le nuove informazioni riguardanti il luogo da raggiungere e, una volta individuata l'ambulanza (passaggio sucessivo), può essere aggiornato anche lo stato in "Moving to Patient". Supponiamo che il processo di shadowing avvenga attraverso un device di gestione utilizzando le API dell'HTTP Adapter. Esso effettuerà la seguente richiesta:

   PUT http://dominioshadow/api/twins/mission/Mission1

   con il seguente body:

   ```
   {
       "status" : 1,
       "place" : "Via Test, 8, XYXYXY Test XX"
   }
   ```

   Listato 7.5: Body della richiesta di aggiornamento missione

2. Individuazione ambulanza e collegamento: considerando che i Digital Twins delle ambulanze disponibili siano già stati creati, in quanto sono asset pre esistenti rispetto alla missione, è solamente necessario individuare l'ambulanza più vicina al luogo della missione e con abbastanza

carburante per supportare il trasporto del paziente. Fatto ciò, l'ambulanza in questione viene collegata alla missione attraverso la seguente richiesta all'HTTP Adapter (perciò processo di shadowing):

PUT http://dominioshadow/api/twins/Mission1/relationships/create

con il seguente body:

```
{
    "relation" : "vehicle",
    "target" : "Ambulance1"
}
```

Listato 7.6: Body della richiesta di creazione relazione tra missione e ambulanza

In questo modo, l'ambulanza può partire e raggiungere il luogo della Missione. A questo punto, avvengono i primi contatti con il paziente interessato e attraverso i device degli operatori dell'ambulanza viene creato il Digital Twin del paziente attraverso la stessa tipologia di richiesta vista per la missione, supponiamo con una fidelity di cinque secondi ed id *Patient1*. Il Paziente è ora diventato ufficialmente parte della missione nel mondo reale, perciò, grazie al processo di shadowing, tutto ciò deve essere replicato anche nel mondo digitale. Quindi, il device dell'operatore attraverso le API, oppure l'ambulanza stessa attraverso i componenti interni, può effettuare richiesta al processo di shadowing per la creazione della relazione tra la missione e il paziente (utilizzando la stessa API di prima, oppure attraverso MQTT nel caso fosse l'ambulanza). Inoltre, nel frattempo, il paziente viene messo in ambulanza dove vengono eseguiti controlli periodici attraverso i macchinari collegati. I macchinari, possono automaticamente inviare, tramite MQTT, gli aggiornamenti riguardanti il paziente rispettando la fidelity. Per emulare questo processo è stato creato un piccolo programma che, utilizzando la libreria *Eclipse Paho*, invia i dati al broker MQTT Mosquitto precedentemente installato. Allo stesso modo, anche l'ambulanza comunicherà attraverso MQTT i propri dati riguardanti la

posizione e il carburante rimanente al fine di mantenere sincronizzata la sua controparte digitale.

Dopo le relazioni create i tre Digital Twins si trovano disposti in un grafo. È possibile vedere il grafo da Azure Digital Twins Explorer nella Figura 7.1:

Figura 7.1: Grafo di Azure Digital Twins

Un agente esterno potrebbe eseguire diverse tipologie di richieste:

- Ottenimento dello stato completo del paziente: è possibile, sfruttando l'API offerta dal nodo, ottenere lo stato attuale del paziente. Per fare ciò basterà effettuare la seguente richiesta:

  GET http://dominio/api/twins/Patient1

  e si riceverà lo stato del Digital Twin direttamente in RDF:

```
@prefix adt:       <http://dominio/api/twins/> .
@prefix adtprops: <http://dominio/properties#> .
@prefix ex:        <http://example.com/> .

adt:Patient1  ex:bloodPressure  "80"^^ <http://www.w3.org/2001/XMLSchema#int> ;
              ex:bodyTemperature "36.5"^^ <http://www.w3.org/2001/XMLSchema#double> ;
              ex:heartRate       "75"^^ <http://www.w3.org/2001/XMLSchema#int> ;
              ex:patientCode     "3"^^ <http://www.w3.org/2001/XMLSchema#int> ;
              adtprops:inSynch   true .
```

Listato 7.7: Stato di un Digital Twin in RDF

Si può notare che vengono specificate tutte le proprietà del Digital Twin attualmente memorizzate e viene anche fornito lo stato di validità attraverso il predicato "adtprops:inSync" che, in questo caso, è impostato a *true*, perciò il Paziente è considerato in uno stato "sincronizzato" rispetto alla controparte fisica.
Nel caso in cui il paziente non riceva alcun dato entro le tempistiche stabilite, il Quality and Fidelity engine imposterebbe uno stato di non validità e la proprietà "adtprops:inSync", in una successiva richiesta, avrebbe valore *false*.

- Esecuzione di una query: un agente potrebbe avere la necessità di eseguire una query in SPARQL al fine di ottenere le informazioni contenute nel grafo di conoscenza interno all'organizzazione. Per fare ciò, sarà sufficiente eseguire la seguente richiesta:

  POST http://dominio/api/twins/sparql

  specificando nel body la query da eseguire. Le due query di esempio descritte in Figura 6.4 e in Figura 6.6 sono state pensate proprio per questo caso di studio.

- Osservazione di un Digital Twin: un'ipotetica interfaccia grafica potrebbe voler mostrare lo stato attuale della missione visualizzando in real-time la posizione dell'ambulanza e lo stato del paziente al suo interno. Per fare ciò basterà eseguire le opportune richieste di osservazione, una per ogni Digital Twin di interesse, e gestire gli eventi ricevuti. Una semplice richiesta per osservare il Digital Twin dell'Ambulanza è:

  GET http://dominio/api/twins/Ambulance1/events

  La richiesta consentirà l'instaurazione di una web socket nel quale si riceveranno tutti gli eventi in cui *Ambulance1* è coinvolta espressi nel seguente formato (nel caso di evento di update):

```
{
    "id" : "Ambulance1",
    "type" : "DigitalTwin.updated",
    "patch" : "JSON-PATCH"
}
```

Listato 7.8: Evento ricevuto nel processo di osservazione

Al posto di "JSON-PATCH" verranno restituiti, nel caso di evento di update, i cambiamenti che ha subito il Digital Twin in formato *Json Patch*, come da requisito.

Si può ben capire come sia possibile creare sopra alle funzionalità offerte una quantità enorme di servizi aggiuntivi che, appoggiandosi all'ecosistema, riescono a fornire informazioni riguardanti la realtà e/o eseguire previsioni (collegando un sistema di intelligenza artificiale al processo di osservazione) che possono aiutare i protagonisti nel mondo reale nel risolvere le situazioni più complesse.

Questa visione, offerta da Web of Digital Twins e proposta in questa architettura, spinge a non pensare solo a livello dell'organizzazione quindi in questo caso solo al medico o all'ospedale che utilizza i suoi Digital Twins. Il vantaggio offerto da questa architettura è che ora chiunque abbia i permessi può ottenere informazioni sulla realtà da qualsiasi organizzazione provenga. Un esempio, che potrebbe contraddistinguere la città del domani, riguarda le Ambulanze interne ad ogni ospedale. Esse potrebbero essere osservate dai Digital Twins o da altri agenti presenti in organizzazioni (quindi nodi) completamente diversi al fine di offrire servizi aggiuntivi. Si pensi ad un sistema di *smart city* che gestisce i Digital Twins dei vari semafori e dei vari incroci presenti all'interno di una determinata città. Non appena viene creata una relazione inter-organizzazionale tra il Digital Twin dell'ambulanza (appartenente all'organizzazione dell'ospedale) e il Digital Twin di un incrocio coinvolto (appartenente all'organizzazione che gestisce il sistema di smart city) oppure nel momento in cui viene aggiornato lo stato dell'ambulanza con la città in

cui deve prestare soccorso, si potrebbero attivare degli agenti che, osservando lo stato attuale, adattano il traffico in modo tale da far trovare all'ambulanza in transito tutti i semafori verdi e che allo stesso tempo segnalano ai Digital Twins dei veicoli la presenza di un veicolo di soccorso nelle vicinanze aumentando l'attenzione prestata alla guida (considerando che anche tutti i veicoli avranno i propri Digital Twin appartenenti alle organizzazioni dei produttori che joinano l'ecosistema distribuito). Questo, potrebbe abilitare una serie di vantaggi molto importanti come la riduzione del numero di incidenti dei veicoli di soccorso, la riduzione del tempo di intervento e una generale maggiore sicurezza stradale grazie al layer applicativo che è possibile sviluppare sopra all'ecosistema.

7.2 Considerazioni

Come visto dal caso di studio, il sistema sviluppato ha già buone potenzialità e ha fatto i primi passi verso il concetto di *Web of Digital Twins*. Tuttavia, sicuramente non è completo e necessita di tantissimo studio e sopratutto di tantissimi sviluppi ulteriori. Infatti, durante la progettazione, sono stati presi in considerazione degli elementi non presentati all'interno della tesi in quanto necessitano ancora di approfondimenti ed esperimenti.

I lavori futuri che sono necessari per avanzare e rendere più completa l'architettura possono essere sintetizzati nel seguente elenco:

- Digital Twins descritti da più modelli: in [18] viene evidenziata la necessità di poter modellare un Digital Twin utilizzando più ontologie. In questo modo sarebbe possibile mostrare solamente le informazioni di rilievo per il caso d'uso dell'agente che le richiede. Le soluzioni esistenti sul mercato ancora non prevedono questa possibilità, perciò è necessario individuare un meccanismo per supportare questa funzionalità.

- Comportamento e augmentation: nella modellazione di un Digital Twin rientra anche l'aspetto compartimentale. Deve essere possibile definire il comportamento di un Digital Twin in modo tale da poter rappresentare

anche tutti i comandi che possono essere inviati a quest'ultimo per eseguire le varie operazioni sui propri asset.
Esistono alcune tecnologie per poter rappresentare il comportamento di un Digital Twin, però, fin'ora, nessuna è predominante. Un primo approccio, implementabile all'interno dell'architettura proposta, potrebbe essere simile a ciò che accade in *Web of Things*[1]. Si potrebbero creare delle proprietà aggiuntive all'interno dei Digital Twins che corrispondono agli endpoint a cui connettersi per eseguire un comando. In questo modo, facendo una semplice richiesta a quell'URL si potrebbe richiederne l'esecuzione:

http://dominio/{id}#comando

A livello di modello occorrerà solamente fornire una descrizione testuale delle sue finalità assieme ad una descrizione formale di tutti i parametri necessari e del risultato eventualmente restituito. L'implementazione di tale comando sarà demandata allo sviluppatore incaricato di importare il modello all'interno dell'organizzazione.

Una volta definiti i comandi si potrebbe pensare di esporre, come per lo stato, diverse interfacce a seconda del caso d'uso dell'agente. Ispirandosi al DTDL di Microsoft potrebbe essere utile consentire l'ereditarietà tra modelli in modo tale da poter definire gerarchie di Digital Twins che ereditano anche le capacità compartimentali. In questo modo, sarebbe possibile definire Digital Twins sempre più specializzati che aggiungono funzionalità al servizio base, consentendo di implementare in miglior modo il concetto di *Servitization* definito da Minerva, Lee e Crespi in [15].

Uno dei problemi che rimane ancora irrisolto è il fatto che ogni Digital Twin può ricevere dati da più sorgenti e quindi, idealmente, un singolo Digital Twin corrisponde all'unione di più asset fisici. Quando, attraverso l'augmentation, vogliamo inviare un comando ad un asset fisico è

[1] `https://www.w3.org/TR/2020/REC-wot-thing-description-20200409/`

necessario un meccanismo per specificare l'asset fisico a cui intendiamo rivolgerci.

- Simulazione: la simulazione è un aspetto ancora poco trattato nei Twin Builder ad oggi presenti sul mercato anche se dovrebbe essere incluso all'interno dell'ecosistema un modo per permettere la simulazione di alcuni fenomeni di interesse.

- Previsione: attualmente, con la soluzione offerta in questa tesi, è possibile agganciare un sistema di previsione e di intelligenza artificiale al processo di osservazione di un Digital Twin. Infatti, osservando e analizzando un Digital Twin possono essere ottenuti i dati necessari per l'addestramento e successivamente, ottenendo gli eventi, sarà possibile passarli al modello al fine di eseguire previsioni di vario tipo. Tuttavia, potrebbe essere utile prevedere un componente interno ad ogni nodo che si occupi di fornire questo servizio tramite augmentation.

- Upload di nuovi modelli: considerando la natura dell'ecosistema, si potrebbe prevedere che particolari agenti, dotati di permessi speciali, possano eseguire l'upload di nuovi modelli per i Digital Twins direttamente da API e quindi senza doverli importare manualmente nel Twin Builder interno al nodo. Eseguire l'upload di un modello, rispettando gli standard, richiederebbe l'utilizzo dei linguaggi OWL e RDFS con eventuale controllo dei dati attraverso SHACL. Dopodiché, ogni modello espresso in OWL ed RDFS dovrebbe essere convertito nel linguaggio utilizzato internamente dal Twin Builder. Tutto ciò è già realizzabile in quanto, come mostrato nelle parti precedenti della tesi, esistono già convertitori che a partire da un'ontologia OWL/RDFS restituiscono un modello espresso in DTDL di Microsoft, quindi sviluppare convertitori ad-hoc non dovrebbe essere un problema.
  Il problema, tutt'ora da risolvere, è come far fronte alla creazione delle classi che effettuano il mapping del modello in oggetti all'interno del middleware. Si potrebbe pensare a due approcci principali. Il primo

sarebbe la creazione di classi generali, capaci di adattarsi a qualsiasi contesto. Il secondo coinvolgerebbe l'utilizzo di una tecnica chiamata *Riflessione* (Reflection).
Occorre ancora eseguire studi ed esperimenti per arrivare ad una soluzione che soddisfi tutti i requisiti.

- Query distribuite: ispirandoci a [18] abbiamo individuato la necessità degli agenti di poter definire query nel distribuito che riguardano le varie organizzazioni. In questa tesi, si è fornito un primo approccio a questa necessità che però richiede migliorie in quanto non del tutto aderente con i nostri obiettivi per il futuro. Infatti, la presenza di due punti diversi in cui eseguire le query (nodo e core) non rappresenta a pieno il concetto.
  Tuttavia, abbiamo anche capito che poter definire una query distribuita significa definire in modo ottimale il concetto di *tempo*. Infatti, è necessario far si che la query ottenga dati consistenti e soprattutto che in tutte le organizzazioni si possa far vale lo stesso concetto di *istante* e di *tempo*.

  I termini "ora" e "istante" in un sistema distribuito a cosa corrispondono?

  Nel tempo si sono individuati due modi principali: le *transazioni* e gli *snapshot*. Le *transazioni* bloccano tutto l'ecosistema per poter eseguire una singola query. In questo modo si è sicuri della consistenza della query stessa anche se è impensabile adottare un modello transazionale in un tale ecosistema. Infatti, ci sono processi, come quello di shadowing, che non possono essere "semplicemente" messi in pausa in quanto si andrebbe proprio contro al principio di Digital Twin o Digital replica. Una soluzione migliore da questo punto di vista è lo *snapshot* anche se presenta ulteriori problemi dal punto di vista del tempo in cui questo snapshot viene eseguito.

  Si capisce quindi che le query distribuite sono un problema ancora ad oggi difficile a livello di ricerca in quanto l'assunzione che tutto sia "presente" e "attuale" in un ambiente distribuito non è sempre valida. Basti pensare

che un Digital Twin ha una data di ultimo aggiornamento quindi, da un certo punto di vista, potrebbero essere i Digital Twins stessi a dirci quali istanti temporali abbiano a disposizione.

- Query sulla storia: permettere agli agenti e ai componenti dei nodi interni di ottenere snapshot dell'ecosistema o di una sua parte in un determinato momento storico.

- Comunicazione tra Digital Twins: i Digital Twins, nel caso di *Cognitive Digital Twins*, potrebbero avere la necessità di comunicare tra loro o di richiedere l'esecuzione di azioni.

- Autenticazione e autorizzazione: è necessario un meccanismo con cui gli agenti si autentichino all'ecosistema e anche un sistema di autorizzazione che definisca i limiti di ciascun agente. Un agente potrebbe essere abilitato all'esplorazione di solo un sottoinsieme dell'ecosistema o delle sue funzionalità.
  Inoltre, è necessario prevedere un sistema di autenticazione e autorizzazione anche per i nodi che si collegano al core dell'ecosistema. Oltre a ciò, è ugualmente importante prevedere un handshake tra il nodo e il core seguito da un processo di sincronizzazione, anche solo, banalmente, riguardante gli stati di validità dei Digital Twins interni ad un nodo.

- Qualità e fidelity estesa: come accennato precedentemente, potrebbe essere necessario stabilire dati più specifici riguardanti i requisiti temporali di ogni tipo di Digital Twin. Quindi anche a livello della singola proprietà o della singola relazione, fornendo così informazioni più precise all'agente che, in questo modo, può sapere quale parte del Digital Twin è fuori sincronizzazione e quindi da considerare come non valida.

- Ownership: come previsto da altre architetture, potrebbe essere utile prevedere un meccanismo per garantire la validità e la disponibilità delle informazioni basilari sui proprietari degli asset e dei Digital Twins associati.

Conclusioni

Il middleware e l'architettura sviluppata nascono dall'unione di diverse influenze provenienti dai principali approcci, ad oggi conosciuti, alla tecnologia *Digital Twins*.

La visione adottata è nell'ottica di *Web of Digital Twins* [18] la quale ha l'obiettivo di creare un sistema distribuito, event-driven, decentralizzato ed interoperabile che consente il link e l'utilizzo di Digital Twins appartenenti ad organizzazioni diverse.

La proposta descritta in questa tesi ha permesso di fare i primi passi ed avere i primi risultati in questo campo complicato. Ha permesso di implementare parecchie funzionalità e soprattutto di fornire un primo approccio all'interoperabilità tra Digital Twins. Le potenzialità di questo sistema sono già abbastanza alte. Infatti, come si può vedere anche dal caso di studio, grazie al layer applicativo è possibile creare servizi che spaziano su più settori di rilievo per la società.

Il contributo che, credo, sia più importante all'interno di questa tesi è la proposta di architettura. Questa pone le basi teoriche e i requisiti ad un livello di dettaglio tale che l'implementazione può essere eseguita seguendo esattamente le specifiche. Però, il tutto è stato descritto in modo tale da essere agnostici da particolari tecnologie o linguaggi programmazione. L'implementazione presentata è solamente una delle tante possibili ed è servita per dimostrare l'applicabilità della soluzione proposta e le potenzialità riguardo un caso di studio.

Sono molto soddisfatto del risultato ottenuto soprattutto perché credo in questo progetto e in questa visione e spero che un giorno, non troppo lontano,

ogni nazione possa implementare il proprio *Web of Digital Twins* in modo da poterli connettere e formare un sistema a livello globale che possa diventare la vera *replica digitale del mondo*.

Ringraziamenti

Questa tesi è stato un viaggio che mi ha concesso di esplorare argomenti e situazioni a me totalmente nuovi. Ho avuto il piacere di portare avanti e cercare di dare il mio contributo ad un così elegante argomento di ricerca, il tutto con l'opportunità di confrontarmi con ricercatori importanti all'interno dell'università. Vorrei dedicare un ringraziamento particolare al Prof. Alessandro Ricci, relatore di questa tesi, per avermi dato l'opportunità di provare a contribuire a questo argomento, conoscendo anche altri suoi colleghi, e per la disponibilità dimostrata durante tutte le fasi realizzative. Inoltre, grazie anche al Prof. Angelo Croatti, correlatore di questa tesi, e a tutti i componenti del team per i preziosi consigli e per la disponibilità durante tutto questo periodo. Lavorare all'interno di questo gruppo è stato bello e piacevole grazie al clima sereno e aperto che lo contraddistingue.

Questi tre anni di università sono stati intensi e ricchi di emozioni. Mi hanno permesso di affrontare e superare notevoli sfide personali e non. Vorrei ringraziare con tutto il cuore la mia famiglia, in particolare mia mamma, mio babbo e mio fratello che mi sono stati vicini in ogni singolo momento del mio percorso e mi hanno sempre sostenuto, incoraggiato e spronato a dare sempre il massimo, compatibilmente con i miei desideri e le mie aspirazioni. Senza di loro tutto ciò non sarebbe stato possibile e approfitto anche di questa parte della tesi per sottolineare quanto mi senta fortunato ad averli nella mia vita, gli voglio un bene dell'anima e devono sapere e ricordarsi che sono le persone più importanti nella mia vita.

Un ringraziamento va anche a tutti i miei amici che mi hanno supportato e sopportato durante questi anni e anche a tutte le persone che sono andate via,

ho imparato anche da loro. Un ringraziamento speciale però, va a Marco, a Maria e a Manuel con cui ho condiviso ogni singola gioia e dolore di questo percorso. Ci siamo spronati a vicenda per dare il massimo in ogni situazione e spero che lo continueremo a fare anche in futuro, anche se le strade ci porteranno in luoghi diversi.

Ci sono state persone che mi hanno fatto affacciare e credere in questo bellissimo mondo. Innanzitutto mio babbo quando vide la mia passione per i computer e, una sera in cantina, mi disse di volermene comprare uno tutto per me per fare i miei "esperimenti". La mia famiglia mi ha sempre spinto a non smettere mai di credere nei miei sogni e continua tutt'ora a dirmi ogni giorno di non mollare. Anche alcuni professori delle superiori sono stati di fondamentale importanza per tutto ciò. Vorrei ringraziare la mia Prof. Francesca, per esserci sempre stata sia a scuola che fuori e di non avere mai perso contatti anche dopo il diploma. Inoltre, vorrei ricordare la persona che mi ha presentato l'indirizzo informatica alle superiori e con cui ho avuto rapporti bellissimi durante le lezioni, il Prof. Claudio a cui dedico un ricordo particolare e un sorriso da quaggiù.

Grazie a tutti, è stato bello e spero che in futuro arriveranno altre soddisfazioni come questa.

Bibliografia

[1] OWL 2 web ontology language document overview (second edition). Technical report, W3C, December 2012. https://www.w3.org/TR/2012/REC-owl2-overview-20121211/.

[2] SPARQL 1.1 overview. W3C recommendation, W3C, March 2013. https://www.w3.org/TR/2013/REC-sparql11-overview-20130321/.

[3] Dan Brickley and Ramanathan Guha. RDF schema 1.1. W3C recommendation, W3C, February 2014. https://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

[4] Artem Chebotko, Shiyong Lu, and Farshad Fotouhi. Semantics preserving sparql-to-sql translation. *Data Knowl. Eng.*, 68(10):973–1000, October 2009.

[5] Richard Cyganiak, Markus Lanthaler, and David Wood. RDF 1.1 concepts and abstract syntax. W3C recommendation, W3C, February 2014. https://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

[6] Violeta Damjanovic-Behrendt and Wernher Behrendt. An open source approach to the design and implementation of digital twins for smart manufacturing. *International Journal of Computer Integrated Manufacturing*, 32(4-5):366–384, 2019.

[7] Deloitte. New technologies case study: Data sharing in infrastructure – a final report for the national infrastructure commission. Novembre 2017.

[8] David Gelernter. *Mirror Worlds or the Day Software Puts the Universe in a Shoebox: How Will It Happen and What It Will Mean*. Oxford University Press, Inc., USA, 1991.

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

[10] Michael Grieves and John Vickers. *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*, pages 85–113. Springer International Publishing, Cham, 2017.

[11] Digital Framework Task Group. *White paper: The Gemini Principles*. Technical report, Centre for Digital Built Britain, 2018. Disponibile a `https://www.cdbb.cam.ac.uk/DFTG/GeminiPrinciples`. Ultimo accesso: 20210803.

[12] Hetherington James and West Matthew. *The Pathway Towards an Information Management Framework: A Commons for a Digital Built Britain*. Technical report, Centre for Digital Built Britain, 2020. Disponibile a `https://doi.org/10.17863/CAM.52659`. Ultimo accesso: 20210804.

[13] Kendall John. *National Digital Twin: Integration Architecture Pattern and Principles*. Technical report, Centre for Digital Built Britain, 2021. Disponibile a `https://doi.org/10.17863/CAM.68207`. Ultimo accesso: 20210806.

[14] Dimitris Kontokostas and Holger Knublauch. Shapes constraint language (SHACL). W3C recommendation, W3C, July 2017. https://www.w3.org/TR/2017/REC-shacl-20170720/.

[15] R Minerva, GM Lee, and N Crespi. Digital twin in the iot context: a survey on technical features, scenarios and architectural models. *Proceedings of the IEEE*, 2020.

[16] Office of National Statistics. Developing new statistics of infrastructure. Agosto 2018.

[17] Marco Picone, Angelo Croatti, Stefano Mariani, Sara Montagna, and Alessandro Ricci. *Event-Driven Digital Twins*. 2021.

[18] Alessandro Ricci, Angelo Croatti, Stefano Mariani, Sara Montagna, and Marco Picone. Web of digital twins - a multiagent systems perspective. *Transaction on Internet Technology*, 2021.

[19] Alessandro Ricci, Michele Piunti, Luca Tummolini, and Cristiano Castelfranchi. The mirror world: Preparing for mixed-reality living. *IEEE Pervasive Computing*, 14(2):60–63, 2015.

[20] Andy Seaborne. SPARQL 1.1 query results JSON format. W3C recommendation, W3C, March 2013. https://www.w3.org/TR/2013/REC-sparql11-results-json-20130321/.