ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

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

UNA PIATTAFORMA PER DIGITAL TWIN BASATA SU KNOWLEDGE GRAPH

Elaborato in
SISTEMI EMBEDDED E INTERNET-OF-THINGS

Relatore
Prof. ALESSANDRO RICCI

Presentata da
RICCARDO PEPE

Corelatore
Dott. SAMUELE BURATTINI
Prof. MARCO PICONE

Anno Accademico 2022 – 2023

"C'è una forza motrice più forte del vapore,
dell'elettricità e dell'energia atomica: la volontà"
Albert Einstein

Indice

Capitolo 1

Introduzione

Negli ultimi anni l'espressione "Digital Twin" (DT) è sempre più sulla bocca delle persone, dal semplice appassionato di informatica ai dirigenti delle multinazionali; basterebbe questo a dare un'idea sull'importanza che sta ricoprendo questa nuova tecnologia nella società, le industrie 4.0 lo utilizzano ampiamente, ma è in crescita anche in altri campi come le smart city, gli ospedali, il settore automobilistico e tanti altri.

Inevitabilmente, con il crescere della popolarità di una tecnologia, crescono gli strumenti a suo supporto e gli studi in merito, sebbene ci sia una mancanza di uniformità nel mercato.

Un'idea nata in questo contesto è il *Web of Digital Twin* (WoDT), che estende l'idea di DT da entità singola strettamente legata al dominio di appartenenza, ad un ecosistema di gemelli digitali collegati fra loro.

Una prima implementazione dell'idea di WoDT è rappresentata dalla *WoDT platform*, ovvero un progetto che estende la *WhiteLabel Digital Twin*, un framework open-source per lo sviluppo di DT.

Questo progetto di tesi si pone l'obiettivo di implementare un componente della piattaforma WoDT, ovvero il *Knowledge Graph Engine*, un componente incaricato di memorizzare lo stato dei DT sotto un punto di vista semantico. Questo componente risulta cruciale per poter sfruttare appieno la piattaforma e la rete di DT che si va a creare; una funzionalità che dovrà esporre, infatti, è la possibilità di interrogare questo grafo, ricavando informazioni in modo rapido e completo.

Questo documento è suddiviso in quattro capitoli, nel primo verrà proposta un'introduzione al dominio dei DT e alle tecnologie che verranno adottate in questa tesi: viene proposta una breve introduzione sui *Digital Twin*, con un cenno di storia ed i loro principali utilizzi. Inoltre, per comprendere appieno il funzionamento della *WoDT Platform* è necessario introdurre la WLDT, ovvero il framework per la creazione dei DT estesa dal progetto *WoDT Platform*.

Successivamente si introdurrà il concetto di WoDT e la piattaforma che è stata realizzata su questo concetto. Utile è, infine, accennare la tecnologia dei *knowledge graph*, il web semantico e le sue tecnologie come RDF, OWL e SPARQL, strumenti che verranno utilizzati in questo progetto di tesi.

Nel capitolo due si procederà con la fase di progettazione del *Knowledge Graph Engine*, dove si analizzerà il progetto WoDT e si valuteranno le modifiche da effettuare ai componenti esistenti e nel caso le parti da implementare.

Verrà illustrata poi nel capitolo tre l'implementazione dei componenti analizzati nel capitolo precedente utilizzando le tecnologie adeguate, per poi concludere nel capitolo finale con dei test utili per validare il lavoro svolto ed un'analisi dei possibili sviluppi futuri che riguardano la piattaforma WoDT.

1.1 Digital Twin

In questa sezione verranno introdotti i DT, proponendo una definizione e raccontando brevente la loro storia. Nell'ultima sezione verranno elencati alcuni utilizzi comuni di questa tecnologia.

1.1.1 Definizione di Digital Twin

Un *Digital Twin* (Gemello Digitale) può essere definito come una macchina (fisica o virtuale) o un modello informatico che simula e rispecchia il ciclo di vita di una entità fisica. Ogni Digital Twin (DT) è legato da una funzione biettiva (ovvero di corrispondenza uno a uno) tra il DT e il relativo *Physical Asset* (l'entità fisica). Non solo questo gemello digitale simula il comportamento del gemello fisico, ma ne estende le potenzialità, attraverso modelli intelligenti che si evolvono autonomamente. Infatti il Digital Twin è in grado di predirre malfunzionamenti futuri sull'asset fisico, simulare e testare nuove configurazioni. Un DT comunica in tempo reale con il *Physical Asset* (PA) scambiandosi continuamente informazioni, in modo tale da mantenersi sincronizzati. Questo continuo flusso di informazioni, associato ad altre tecnologie come l'intelligenza artificiale, permette di scoprire nuovi dati sul sistema, pattern nascosti e correlazioni sconociute [4].

1.1.2 Storia dei Digital Twin

Le prime tracce di DT risalgono agli anni '70, quando la NASA creò un sistema virtuale che monitorava oggetti fisicamente irraggiungibili ed in grado di trovare soluzioni ai problemi che si potevano presentare. Questi strumenti permisero di addestrare adeguatamente l'equipaggio delle missioni e fu determinante per la risoluzione di un problema alle riserve di ossigeno nella missione

Apollo 13. Sebbene l'idea di simulare delle entità fisiche in un ambiente virtuale fosse già nota, il concetto di DT venne informalmente condiviso nel 2002 da Michael Grieves durante la sua presentazione sulla " Product Lifecycle Management" (PLM), poi formalizzato in un suo articolo l'anno successivo. Questa prima definizione definiva i DT composti da tre parti: uno spazio contenente l'entità fisica, uno spazio contenente l'entità virtuale e un collegamento bilaterale dove ogni gemello può mandare informazioni all'altro. In quegli anni raccogliere grandi quantità di dati per realizzare un modello virtuale non era semplice a causa delle tecnologie ancore immature. Con l'evoluzione di alcuni campi dell'informatica, come IoT (Internet of Things), Big Data e l'Intelligenza Artificiale, la popolarità dei DT è aumentata esponenzialmente [4].

1.1.3 Principali utilizzi

I campi di applicazione dei Digital Twins sono svariati e in continuo aumento, ecco alcuni esempi:

- **Sanità**: nel mondo della sanità i DT sono stati prima impiegati per la manutenzione dei dispositivi medici, e negli ultimi anni sono stati impiegati per ottimizzare l'uso delle risorse e la loro gestione all'interno degli ospedali [4].
- **Industria 4.0**: i DT per la digitalizzazione delle industrie si è rivelato il punto di svolta per l'ottimizzazione del processo di produzione del prodotto e il suo ciclo di vita [4]
- **Edifici Smart**: l'uso di modelli virtuali per supportare il ciclo di vita degli edifici ha iniziato a diffondersi sin dagli anni '90, queste rappresentazioni, però, fornivano dati statici. L'unione negli ultimi anni di queste tecnologie con i DT ha permesso la realizzazione di modelli dinamici che si aggiornano in tempo reale, e che hanno migliorato la gestione della fase di costruzione, delle valutazioni energetiche e termiche degli edifici [6].
- **Città Smart**: la sempre maggiore disponibilità di dati raccolti dalle città (e.g. traffico, consumo elettrico, gestione dei rifiuti...) ha permesso la modellazione di DT d'intere città a supporto della pianificazione urbana e della gestione delle policy [19].
- **Aviazione**: i DT sono molto utilizzati nell'ambito dell'aviazione, sia per una questione di ottimizzazione dei costi e delle tempistiche di manutenzione, ma anche per una questione di sicurezza, cercando di predire guasti e malfunzionamenti [4].

- **Veicoli smart**: il mondo dell'automobile dai DT acquisisce vantaggi non solo sull'aspetto della produzione, ma anche nella sempre più popolare guida autonoma e assistenza alla guida.

1.2 WhiteLabel Digital Twin Framework

La mancanza di standard per lo sviluppo e progettazione di DTs ha portato alla distribuzione di svariati framework (come MS Azure Digital Twins, Oracle Digital Twins ecc...) molto diversi fra loro. Nascono anche progetti open-source, come il progetto Eclipse Ditto, ma carenti sotto il profilo della flessibilità e modularità. Per provare a fornire un framework open-source che sia in grado di fornire una soluzione semplice, estendibile e portabile, è stato sviluppato un framework in Java per lo sviluppo di DT, chiamato WhiteLabel Digital Twin (WLDT).

Un DT è implementato attraverso un software ad agente che implementa tutte le caratteristiche e le funzionalità della sua controparte fisica, a cui può essere collegato per mantenerlo aggiornato nello stato e mantenere il suo ruolo di replica. Ogni DT è dotato di un modello interno, che definisce la rappresentazione digitale dell'entità fisica. Questo modello è denominato *Digital Twin State*, ed è composto da i seguenti componenti:

- **Proprietà**: rappresentano gli attributi osservabili del PA, mappati come valori che possono cambiare dinamicamente con il tempo.
- **Eventi**: rappresentano gli eventi a livello di dominio che possono essere osservati.
- **Relazioni**: rappresentano i collegamenti esistenti fra il PA modellato e altri PA.
- **Azioni**: rappresentano le azioni che possono essere richiamate dal PA attraverso l'interazione con il DT.

Un concetto molto importante per un DT è il processo di shadowing, ovvero l'operazione che permette di mantenere sincronizzato il Digital Twin State con la rispettiva risorsa fisica. Lo *shadowing* è un concetto legato al ciclo di vita del DT ed in questo framework è composto da 5 stati (rappresentati in figura 1.1), seguendo le indicazioni della letteratura scientifica dalla creazione alla conclusione, ovvero:

- **Operating & Not Bound**: è lo stato in cui il DT si trova appena dopo l'inizializzazione dello stesso, ovvero tutti i moduli del DT sono attivi ma non c'è alcun tipo di associazione con il PA.

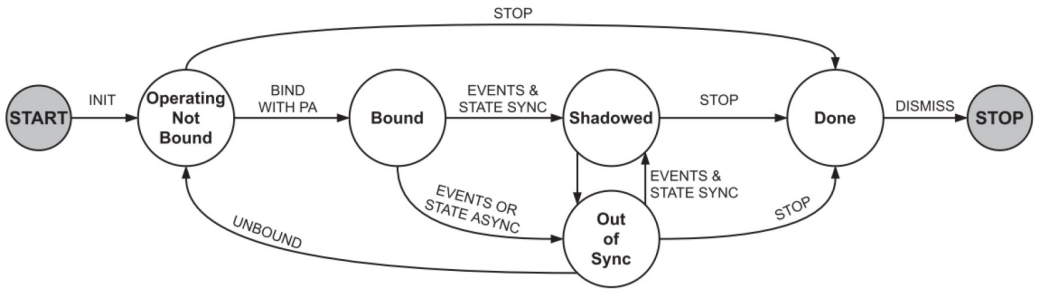

Figura 1.1: Ciclo di vita di un DT. Immagine tratta da [16]

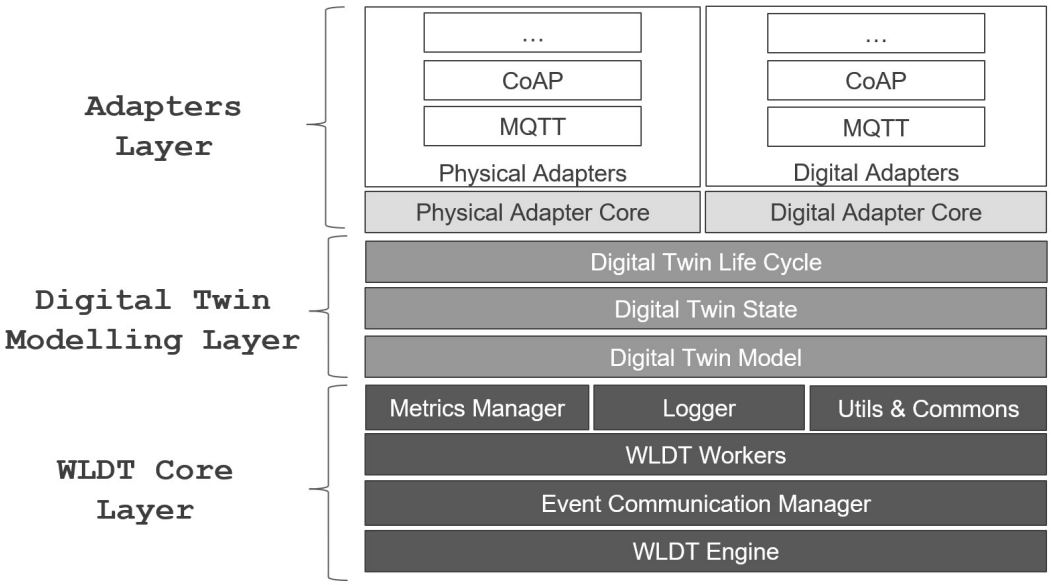

Figura 1.2: Struttura WLDT. Immagine tratta da [16]

- **Bound**: è lo stato che segue l'esecuzione della procedura di collegamento
- **Shadowed**: è lo stato raggiunto dal DT quando inizia il processo di shadowing ed è correttamente sincronizzato con il PA
- **Out of Sync**: è lo stato che determina la presenza di errori nel processo di shadowing
- **Done**: è lo stato che raggiunge il DT dopo che viene stoppato il processo di shadowing

WLDT è un software che implementa tutte le funzionalità di un Digital Twin che lavora su un cloud o su un sistema distribuito. L'architettura del fra-

mework è su layer, come rappresentato in figura 1.2. Questi sono i componenti principali:

- **WLDT Engine**, un sistema multithread capace di gestire svariati workers e i vari moduli interni all'architettura.
- **WLDT Event Bus**, un bus interno progettato per supportare le comunicazioni fra i componenti di una istanza del DT.
- **WLDT Workers**, modellano il componente interno base e costituisce l'elemento eseguibile del WLDT Engine.
- **Digital Twin State**, costituisce lo stato del DT, su un elenco di proprietà, eventi, relazioni ed azioni.
- **Shadowing Model Function**, è il componente della libreria che si occupa di definire il comportamento del DT interagendo con il Digital Twin State.
- **Physical Adapter**, definisce le funzionalità essenziali che il DT dovrà implementare. Un DT può implementare più Physical Adapter, in modo da poter comunicare con la corrispondente entità fisica. Ogni Physical Adapter produrrà un Physical Asset Description (PAD), ovvero una descrizione delle proprietà, eventi, azioni e relazioni che l'entità fisica espone attraverso un determinato protocollo.
- **Digital Adapter**, definisce una serie di callback che permettono di modificare il Digital Twin State. Inoltre, un Digital Twin può definire più Digital Adapter per esporre il suo stato e funzionalità attraverso diversi protocolli di comunicazione. [16]

1.3 Web of Digital Twin

In questa sezione verrà illustrato il concetto di WoDT e la piattaforma in cui verrà implementato il *Knowledge Graph Engine*, ovvero la *WoDT Platform*, illustrandone l'architettura.

1.3.1 Introduzione al WoDT

Ciò che ha caratterizzato lo sviluppo dei DT fino a oggi è la loro tipica rappresentazione individualistica e specifica. La realtà che ci circonda è molto più complessa e prevede interconnessioni fra entità fisiche diverse, spesso con domini diversi e organizzazioni diverse. Quello a cui punta l'idea di Web

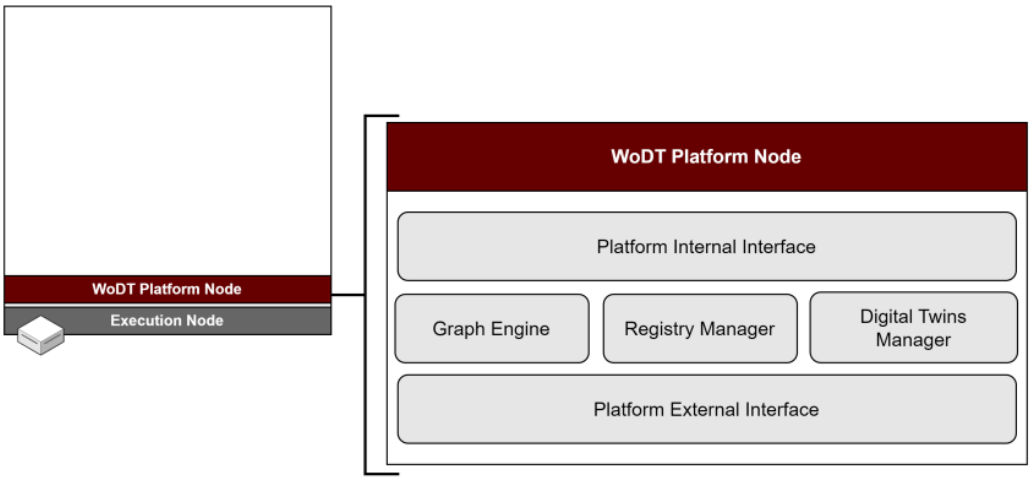

Figura 1.3: Struttura del WoDT Node. Immagine tratta da [18]

of Digital Twin (WoDT) è realizzare un ecosistema di DT interconnessi fra loro, e non concentrarsi più su uno sviluppo tipicamente verticale e specifico, ma allargandolo e mettendo a disposizione a livello applicativo in base alla necessità[15].

1.3.2 WoDT Platform

La *WoDT Platform* è stata realizzata estendendo il framework WLDT in ottica WoDT. La piattaforma è nata con la necessità, data la definizione di WoDT, di supportare sistemi distribuiti, di conseguenza è stato modellato il concetto di nodo della piattaforma, ovvero il WoDT Node. Quest'ultimo presenta un'architettura modulare, come rappresentato in figura 1.3. I componenti principali del Node sono:

- **Platform External Interface**, è l'interfaccia di comunicazione fra il WoDT Node ed il livello applicativo, quindi in grado di ricevere richieste esterni all'ecosistema. È realizzata come un server HTTP REST, e smista le richieste pervenute ai vari componenti del Node.

- **Platform Internal Interface**, è l'interfaccia di comunicazione fra l'ecosistema e i singoli DT. Anche questa componente è realizzata tramite un servizio HTTP REST.

- **Digital Twins Manager**, è il componente che si occupa di gestire il ciclo di vita dei DT.

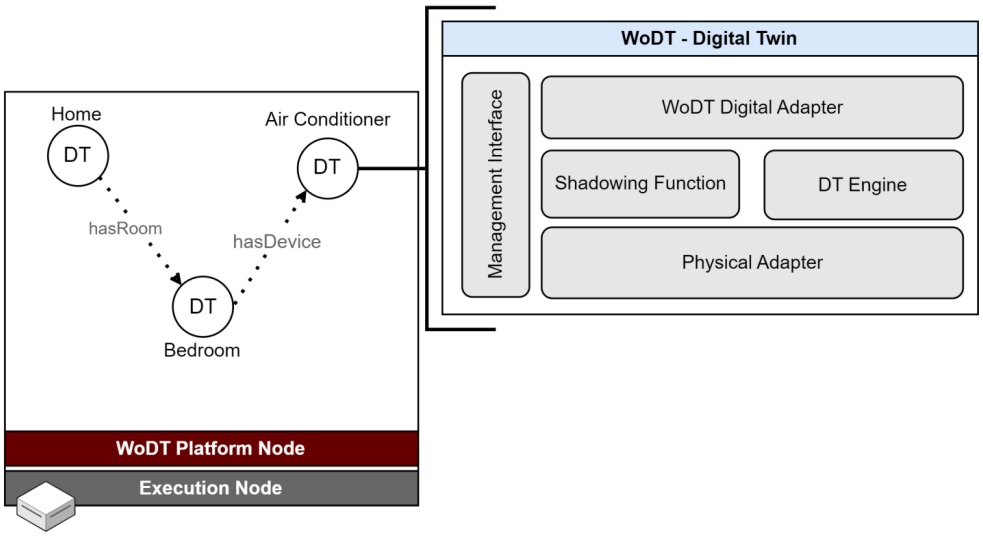

Figura 1.4: Struttura dei DT con modulo *Management Interface*. Immagine tratta da [18]

- **Registry Manager**, è un registro che si occupa di memorizzare l'elenco dei DT inseriti nell'ecosistema ed i loro metadati.
- **Graph Engine**, è il componente che gestisce il grafo dei DT registrati e permette la navigazione dell'ecosistema. Questo componente non è stato ancora implementato nella sua interezza, e questa tesi nasce per dare un'idea di realizzazione di questo componente.

Il WoDT Node permette la centralizzazione dei DT, però è necessario adattare i DT con un modulo che permetta di mettere in comunicazione il DT creato con il framework WLDT, con il WoDT Node; questo modulo è il *Management Interface*, che permette di notificare alla piattaforma il corretto inserimento nell'ecosistema, lo stato aggiornato e la creazione e rimozione delle istanze delle relazioni. La *Management Interface* è realizzata, come un client HTTP [18].

1.4 Knowledge Graphs e Semantic Web

Prima di procede con lo sviluppo del progetto di tesi, viene proposta in questa sezione un introduzione ai knowledge graph, anche in ambito web semantico, per poi proporre le tecnologie che sono state rese uno standard in questo campo, partendo dal Resource Description Framework (RDF) per la rappresentazione semantica, passando per il Web Ontology Language (OWL) che

permette di descrivere un'ontologia, al linguaggio dedicato alle interrogazioni semantiche SPARQL Protocol and RDF Query Language (SPARQL).

1.4.1 Knowledge Graph

Sebbene il termine "Knowledge-Graph" esista in letteratura almeno dal 1972, divenne popolare solamente a partire dal 2012, ovvero all'annuncio del Google Knowledge Graph. Da lì, dato il suo crescente utilizzo a livello industriale, fu analizzato dalla comunità scientifica. I knowledge graph sono dei grafici utilizzati per rappresentare la conoscenza in scenari di applicazione che richiedono l'integrazione, la gestione e l'astrazione dei dati da diverse sorgenti su larga scala (esempio di knowledge graph in figura 1.5).

A causa della grande diffusione di modelli di grafi, algoritmi e i svariati domini d'applicazione, sono stati definite diverse tipologie di database, sistemi e linguaggi per le interrogazioni per grafi. Le tipologie di database utilizzati nei knowledge graph sono i seguenti:

- **Database relazionali**: questa tipologia di database è estremamente efficace nelle interrogazioni, inserimento, rimozione ed aggiornamento delle tavole del database.

- **Database chiave/valore**: la tecnologia chiave/valore è la base dei database NoSQL, questa ottimizza la scalabilità, è ottima per funzionare su sistemi distribuiti su larga scala e semplifica i modelli relazionali tradizionali.

- **Triple store**: è una tipologia di database che si occupa di memorizzare e ricercare triple, ovvero entità di dati composte da soggetto-predicato-oggetto (es. "Marco possiede un'auto"), ed particolarmente indicata per memorizzare informazioni di tipo semantico; questa tecnologia verrà approfondita nelle prossime sezioni.

- **Database map/reduce**: il paradigma map/reduce può essere impiegato nei database per processare in modo efficiente grandi quantità di dati attraverso il calcolo parallelo [20].

Uno strumento molto utilizzato per la modellazione di Knowledge Graph della tipologia *triple store* è la tecnologia RDF, infatti con questa tecnologia è possibile rappresentare i nodi e i collegamenti del grafo attraverso le triple, ma, per poter utilizzare in modo utile il contenuto di un grafo della conoscenza, è necessario dare un significato ai termini utilizzati, quindi è necessario specificare un'ontologia attraverso, ad esempio, il linguaggio OWL (*Web Ontology*

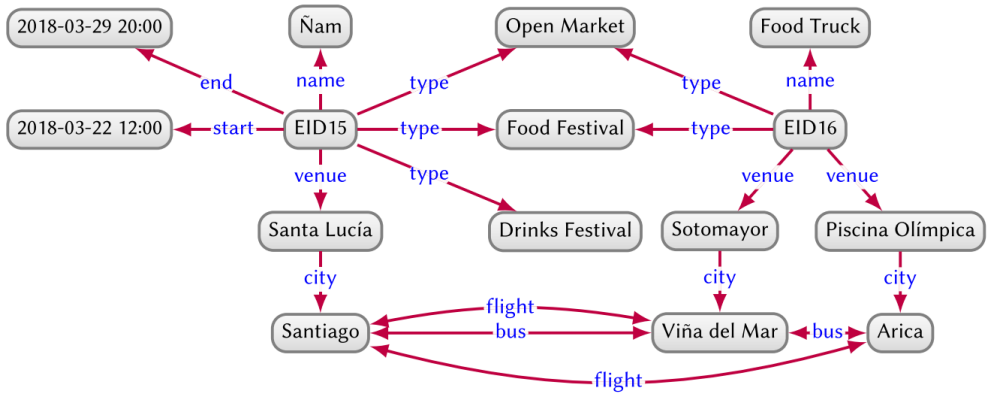

Figura 1.5: Esempio di Knowledge Graph. Immagine tratta da [12]

Language). Inoltre, per poter sfruttare appieno le potenzialita di un knowledge graph è necessario un linguaggio che possa interrogare queste tipologie d'informazioni; nel caso, ad esempio, di un KG basato su una tecnologia RDF, SPARQL risulta una opzione valida a tal fine [12].

1.4.2 Web Semantico

I computer sono nati originariamente con l'idea di aiutare l'uomo con il calcolo numerico, oggi la loro funzione si è ampliata enormemente ed è diventata predominante la sua funzione informativa. Si è reso necessario dare uno strumento all'informatica in grado di poter utilizzare le informazioni sul piano semantico.

Nel 2001 nacque l'idea di web semantico, ovvero un nuovo modo di strutturare i dati incentrato sul significato delle informazioni, in modo tale da poter aumentare la cooperazione fra macchina e utente [5]. Il W3C (World Wide Web Consortium) decise di renderlo uno standard, anche sotto l'influenza di Tim Berners-Lee, che se ne fece promotore [2]. Il W3C rese standard una serie di strumenti in questo campo, per esempio nel 2004 il W3C rese il Web Ontology Language (OWL) uno standard per la descrizione della ontologia(revisionato nel 2012 nel OWL 2). Sempre nel 2004 il Resource Description Frameork (RDF) diventò uno standard W3C, il quale rappresenta un modello utile per rappresentare risorse e le relazioni tra esse attraverso una sintassi XML (eXtensible Markup Language). In sostanza, un'ontologia OWL può fungere da schema per un grafo RDF. Nel 2008 invece venne reso uno standard dal W3C un linguaggio per le query RDF, ovvero SPARQL Protocol and RDF Query Language (SPARQL) (aggiornato con una nuova versione nel 2013).

In seguito, altre tecnologie furono sviluppate e sono ancora in sviluppo in ambito semantic web dal W3C (es. JSON-LD, RDFS, ecc..) [10] [2].

1.4.3 Resource Description Framework (RDF)

Il Resource Description Framwork è un costrutto che permette l'encoding, lo scambio e il riuso di metadati strutturati. Gli RDF sono basati su una sintassi XML, dove però sono imposte delle regole strutturali per poter esprimere la semantica in un modo non ambiguo. Un'ottima caratteristica dei dati RDF è che possono essere letti correttamente sia da una macchina sia dall'uomo. L'uso più comune degli RDF è per descrivere risorse, queste sono identificate unicamente dall'Uniform Resource Identifier (URI). Le proprietà associate alle risorse sono identificate da una "property-type", e a ogni property-type corrispondono dei valori. Negli RDF questi valori possono essere sia atomici (es. stringhe, numeri, ecc...) sia altre risorse.[13] [2]

```
<?xml version="1.0"?>
<rdf:RDF
     xmlns:ns="http://www.w3.org/2006/vcard/ns#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

    <ns:Individual rdf:about="http://example.org/Bianchi">
        <ns:fn>Mario Bianchi</ns:fn>
        <ns:language>Italian</ns:language>
    </ns:Individual>
</rdf:RDF>
```

Listato 1.1: "Esempio di risorsa RDF"

In questo esempio possiamo notare la classica struttura XML, dove il namespace rimanda a un documento esterno che dichiara la semantica e le convenzioni che andremo ad utilizzare per le nostre risorse. [13] In questo caso, l'ontologia scelta è vCard, ovvero una specifica sviluppata dal IETF (Internet Engineering Task Force) per descrivere le persone e le organizzazioni [14]. L'attributo `rdf:about` definisce il nostro URI, ovvero l'identificatore della nostra risorsa `ns:Individual`. Inoltre il nostro individuo avrà due proprietà, ovvero `ns:fn` che corrisponde al nome completo e `ns:language` che indica il linguaggio da dover utilizzare per comunicare con questo individuo [14]. Questo meccanismo permette il riuso delle risorse da parte di sviluppatori che decidono di utilizzare o estendere la stessa ontologia [2].

1.4.4 Web Ontology Language (OWL)

Web Ontology Language (OWL) è un linguaggio che permette di esprimere delle ontologie. Con il termine ontologia, nel campo dell'informatica, si intende un qualsiasi tipo di artefatto computazionale che fornisce una descrizione di una qualsiasi entità (detta "dominio di interesse"). Gli strumenti che OWL fornisce per poter descrivere una ontologia sono le classi, le proprietà, gli individui e i datatype. Dato che OWL è un linguaggio usato nel semantic web, le ontologie sono denominate attraverso degli International Resource Identifiers (IRIs)[11].

La sintassi OWL può essere di svariate tipologie:

- Sintassi funzionale

```
SubClassOf( :Woman :Person )
```

- Sintassi RDF/XML

```
<owl:Class rdf:about="Woman">
    <rdfs:subClassOf rdf:resource="Person"/>
</owl:Class>
```

- Sintassi turtle

```
:Woman rdfs:subClassOf :Person .
```

- Sintassi Manchester

```
Class: Woman
    SubClassOf: Person
```

- Sintassi OWL/XML

```
<SubClassOf>
    <Class IRI="Woman"/>
    <Class IRI="Person"/>
</SubClassOf>
```

Sebbene gli RDF e gli RDFS (RDF Schema) permettano di esprimere alcune ontologie, essi hanno alcune limitazioni:

- **Scope locale delle proprietà**: ovvero attraverso un RDFS non possiamo dichiarare dei range da applicare ad alcune classi limitate (es. possiamo definire la proprietà "mangia" alla classe animale, ma non possiamo dire che alcuni animali mangiano solo piante e altri solo carne).
- **Disgiunzione delle classi**: potrebbe sorgere la necessità di dover esprimere la disgiunzione tra diverse classi.
- **Combinazione booleana di classi**: potrebbe sorgere la necessità di esprimere nuove classi in funzione di classi esistenti attraverso una loro combinazione booleana usando l'unione, l'intersezione e la complementarietà.
- **Restrizioni di cardinalità**: potrebbe sorgere la necessità d'imporre restrizioni su quanti valori distinti di una proprietà vorremmo.
- **Caratterstiche speciali delle proprietà**: a volte potrebbe essere utile esplicitare se una proprietà è transitiva, unica o inversa di un'altra proprietà.

Queste caratteristiche sono state implementate nel linguaggio OWL, permettendo di avere un linguaggio molto più espressivo rispetto al RDFS. Inoltre esistono tre versioni di OWL, in base al livello di espressività desiderato

- **OWL Full**
- **OWL DL**
- **OWL Lite**

La scelta va effettuata in base ad un "trade-off" tra facilità di implementazione (verso OWL Lite) e capicità di espressione e completa compatibilità con RDF/RDFS (verso OWL Full). [1]

1.4.5 SPARQL Protocol and RDF Query Language (SPARQL)

Dopo il rilascio della tecnologia RDF si è subito iniziato a lavorare su un linguaggio che permettesse di effettuare delle query, ovvero SPARQL che divenne standard W3C nel gennaio 2008 con una prima versione e poi espansa con la versione 1.1 diventata raccomandazione W3C nel 2013. Questa espansione aggiunse nuove funzioni e la possibilità di aggiornare i dati, cosa che non era possibile con la versione 1.0 [7]. SPARQL è un linguaggio di query a livello RDF, infatti esistevano già linguaggi per interrogare documenti XML, ma, il

loro uso in un contesto semantico, lavorando ad un livello di astrazione più basso poteva indurre a complicazioni.

```
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>

SELECT ?givenName ?familyName
WHERE {
  ?person a vcard:Individual ;
          vcard:fn ?fullName .

  ?fullName vcard:given-name ?givenName ;
            vcard:family-name ?familyName .
}
```

Listato 1.2: Esempio di query SPARQL che utilizzando l'ontologia vCard richiede l'elenco dei nomi e cognomi delle persone presenti nel grafo

In questo esempio si può vedere come, così come per i documenti RDF, SPARQL permette di dichiarare dei prefissi per i namespace ed utilizzarli nelle query. Inoltre si può notare ricalchi la struttura SELECT-FROM-WHERE tipica di SQL, dove però FROM specifica la sorgente che deve essere interrogata, ed in SPARQL è un parametro opzionale [2].

1.5 Grafi di Digital Twins: tecnologie esistenti

Nei diversi framework esistenti, non sono molti quelli che implementano un knowledge graph, tra questi analiziamo due esempi, ovvero Microsoft Azure Digital Twins e AWS IoT TwinMaker. Entrambi supportano la creazione di grafi a supporto dei DT senza dover obbligatoriamente specificare una ontologia, infatti per quanto riguarda MS Azure il grafo è strettamente collegato all'idea di DTDL, ovvero un linguaggio JSON-like che permette di descrivere le diverse tipologie di entità, e quindi di DT; mentre per quanto riguarda AWS IoT TwinMaker il grafo è realizzato ricalcando la struttura dei DT basati sui concetti di entità e componente, dove le entità sono le rappresentazioni digitali dei diversi elementi del DT, mentre i componenti rappresentano il contesto delle entità.

Entrambi i framework supportano un linguaggio per le interrogazioni, per quanto riguarda MS Azure si tratta di un liguaggio di query proprietario, mentre per AWS TwinMaker si tratta di PartiQL, un linguaggio per accedere a dati relazionali, nidificati o semistrutturati; entrambi sono linguaggi SQL-like.

L'obiettivo progettuale di questa tesi è proprio quello di realizzare un modulo per la piattaforma WoDT che implementi un grafo per DT. Nel prossimo

capitolo verrà descritta la progettazione del modulo, a cui seguirà la descrizione dell'implementazione prototipale sviluppata, basata su tecnologie relative al web semantico.

Capitolo 2

Progettazione Knowledge Graph Engine per WoDT

In questo capitolo si vuole analizzare come estendere l'attuale progetto *WoDT Platform* con un *Knowledge Graph Engine*, componente la cui implementazione non è ancora stata del tutto affrontata, e quindi procedere con la progettazione dei componenti necessari. Lo sviluppo di questo componente seguirà diverse fasi: si partirà con una fase di analisi dei requisiti, per poi passare ad una fase di analisi del progetto esistente e quindi di progettazione, per poi modellare uno scenario di riferimento utile per validare il lavoro svolto.

2.1 Analisi dei Requisiti

L'attuale piattaforma WoDT possiede una predisposizione per la realizzazione di un *Knowledge Graph Engine* (KGE), ovvero il componente che si occuperà di mantenere una rappresentazione dei DT usando una tecnologia volta alla semantica; la sua implementazione non è stata ancora affrontata in modo completo. La realizzazione di un grafo per DT permette la navigazione dell'ecosistema sopratutto attraverso la risoluzione di *query*. Quindi ogni DT dovrà essere in grado di poter inviare al *WoDT Node* una sua rappresentazione da dover registrare nel KGE, che a sua volta potrà essere interrogato a livello applicativo. Sarà inoltre importante che i dati vengano creati, modificati e cancellati solo dal DT stesso, in modo tale da non creare problemi di sicurezza e di integrità delle informazioni.

2.2 Progettazione

Sebbene sia già presente una predisposizione nel progetto *WoDT Platform* per un KGE, è necessario apportare alcune modifiche a vari componenti dell'ecosistema. Infatti i DT dovranno essere capaci di inviare alla *WoDT Platform* una loro rappresentazione improntata sulla semantica; a questo scopo, riprendendo la struttura DT illustrata nella sezione dedicata nella sezione WLDT del capitolo precedente, lato Digital Adapter sarà necessario aggiungere un componente, chiamato *Entity Knowledge Graph Rapresentation* che si occuperà di mantenere una rappresentazione del DT. Poiché la scelta della ontologia e l'implementazione del DT possono avvenire in momenti diversi, è importante isolare l'ontologia di un DT dal DT stesso; per questo si è scelto di creare un'interfaccia *Entity Ontology* che non fa altro che mappare le proprietà del DT con le relative proprietà dell'ontologia scelta. Ne risulta una struttura come quella in figura 2.1.

Inevitabilmente andranno modificate anche le interfacce di comunicazione tra il DT e la *WoDT Platform*, sopratutto per quanto riguarda la gestione di una nuova rappresentazione del DT; infatti questa era contenuta nel *Digital Twin Descriptor* come elenco di proprietà, relazioni, eventi ed azioni. Ora bisogna aggiungere all'interno del *DT Descriptor* una rappresentazione del DT semantica, in modo tale che possa essere comunicata al *WoDT Node*. Un'altra modifica necessaria è, oltre a dover gestire il nuovo *DT Descriptor*, implementare una sorta di autenticazione del DT, in modo tale che solo chi ha chiesto la registrazione dello stesso ha la possibilità di apportare modifiche alla sua rappresentazione sul *knowledge graph*, evitando problemi legati alla sicurezza ed alla integrità dei dati.

Il cuore di questo progetto è il KGE, il quale dovrà implementare le interfacce che sono già state predisposte, ovvero il *Digital Twin Graph Listener*, che contiene i metodi necessari per l'aggiornamento dello stato e la creazione e rimozione delle istanze delle relazioni, il *Digital Twin Registration Listener*, che gestisce la registrazione e lo stop del DT, ed infine una nuova interfaccia che si occuperà di aggiungere un *pattern observer* per le interrogazioni del grafo, che verrà chiamato *Digital Twin Request Query Listener* (figura 2.2).

Infine, per poter effettuare delle interrogazioni a livello applicativo è necessario aggiungere il servizio al *Platform External Interface* che richiamerà il *Digital Twin Request Query Listener* nel KGE.

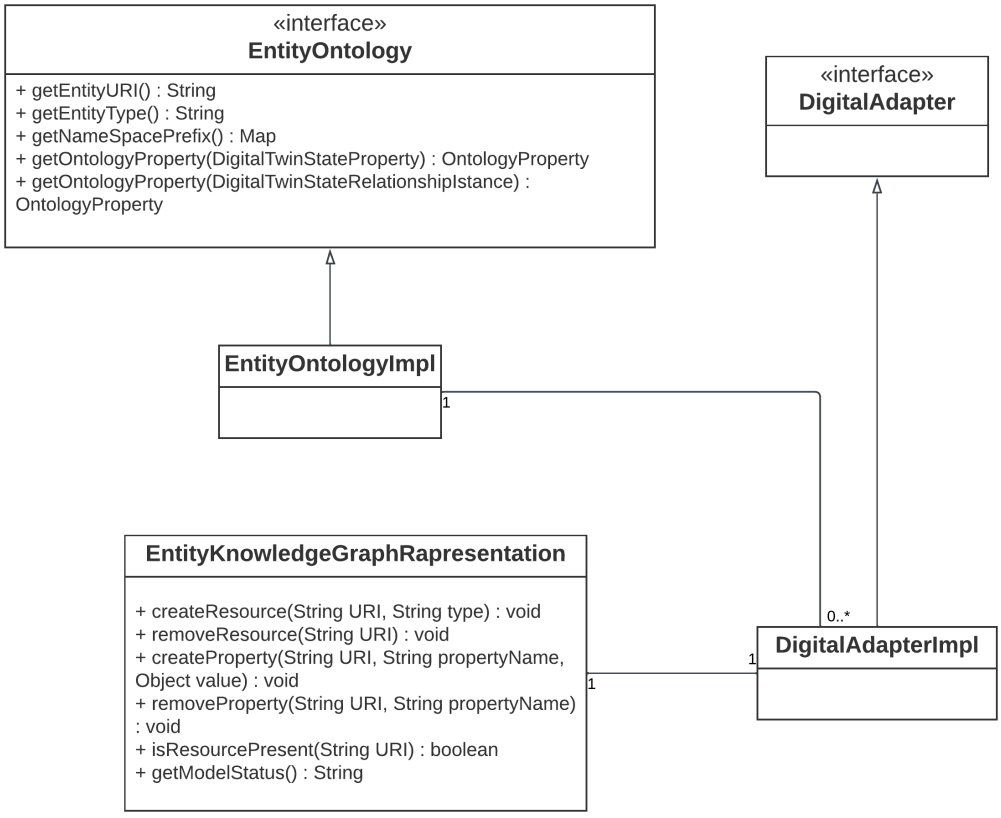

Figura 2.1: Diagramma delle classi UML per i Digital Adapter con rappresentazione ontologica

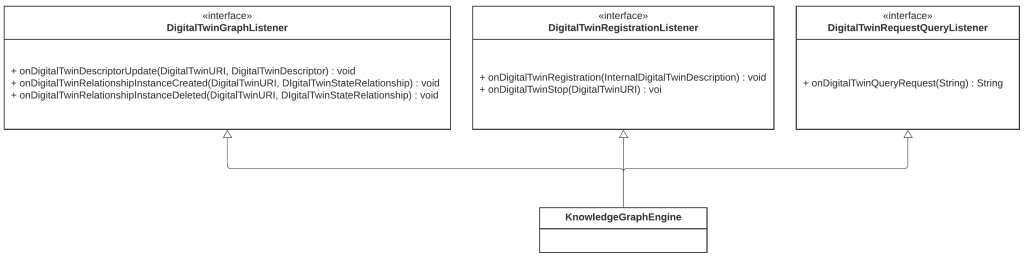

Figura 2.2: Diagramma delle classi UML per il KGE

2.3 Modellazione di uno scenario di riferimento

Per poter testare il progetto è necessaria la realizzazione di DT e quindi è fondamentale predisporre uno scenario di testing. Lo scenario preso in considerazione è quello di un laboratorio all'interno del campus di Cesena, dove, all'interno, sono presenti un punto luce e un rilevatore di movimenti. Il sensore dovrà comunicare se sono presenti movimenti nella stanza e l'istante in cui è stata effettuata la misurazione, invece il punto luce dovrà comunicare il proprio stato al relativo DT. Quindi dovrà essere possibile alla fine della fase di implementazione poter interrogare il KGE per recuperare i dati relativi, ad esempio, allo stato della luce e le misurazioni effettuate dal sensore di movimenti.

Capitolo 3

Implementazione Knowledge Graph Engine per WoDT

3.1 Rappresentazione dello scenario di riferimento usando una ontologia

Il primo passo è ricercare l'ontologia che meglio possa modellare lo scenario di riferimento. Una soluzione arriva dal mondo SAREF (*Smart Applications REFerence*), ovvero una ontologia condivisa proposta da ETSI, ovvero l'istituto europeo per le norme di telecomunicazione. SAREF si concentra nella modellazione dei domini smart. Una delle caratteristiche interessanti di SAREF è la sua estendibilità, difatti sono state pubblicate delle estensioni per specializzare l'ontologia verso determinati domini, per esempio SAREF4AGRI per il dominio dell'agricultura, oppure SAREF4City per il dominio delle città e tante altre [9]. Nel caso dello scenario sopra proposto può risultare particolarmente adatta l'estensione SAREF4BLDG, infatti essa estende SAREF per rappresentare il dominio degli edifici smart.

A questo punto risulta possibile modellare manualmente lo scenario di riferimento utilizzando l'ontologia scelta, in modo tale da avere un punto di riferimento ed una idea del risultato finale da ottenere. Il campus di Cesena può essere rappresentato da un s4bldg:Building, che a sua volta è una specializzazione di geo:SpatialThing, in modo tale da riutilizzare il concetto di locazione proposto dalla ontologia "geo". Il laboratorio invece può essere rappresentato da s4bldg:BuildingSpace che, attraverso la proprietà saref4bldg:hasSpace, si esplicita che il laboratorio è uno spazio dell'edificio. All'interno del laboratorio, attraverso la proprietà saref4bldg:contains, sono presenti un sensore di movimenti ed un punto luce. Il sensore di movimenti effettua una misurazione (saref:Measurement) comprendente: la data e l'orario della misurazione ed il

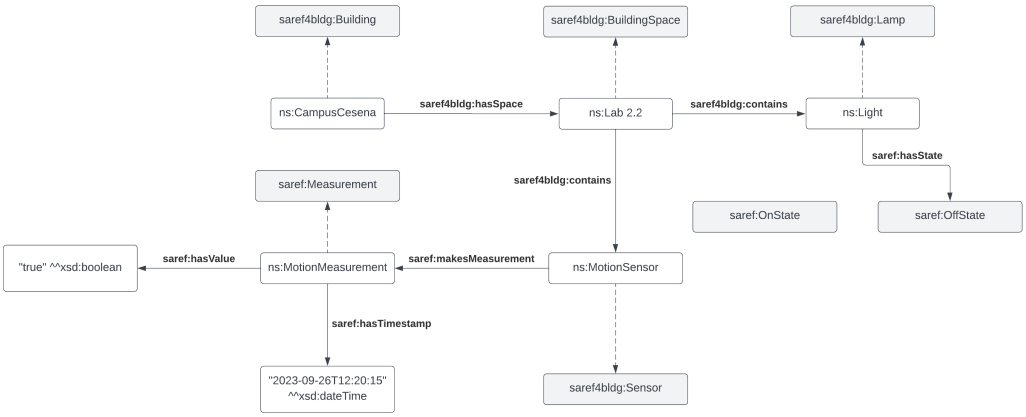

Figura 3.1: Knowledge Graph rappresentate lo scenario di riferimento

valore espresso tramite un valore booleano. Mentre la luce possiede uno stato che può essere Saref:OnState oppure Saref:OffState [8]. La rappresentazione visiva del knowledge graph è quella nella figura 3.1.

3.2 Modellazione dei Digital Twins

Avendo definito un punto di arrivo, è necessario procedere alla progettazione ed implementazione dei DT. La prima fase consiste nell'individuare i DT da realizzare e le loro funzionalità. Analizzando lo scenario di riferimento si può notare come alcuni elementi dello scenario siano statici, come il campus di Cesena oppure il laboratorio, mentre altri risultano attivi, come il sensore di movimenti ed il punto luce. Questi ultimi due elementi invieranno tramite il protocollo MQTT il loro stato al relativo DT. Per quanto riguarda il campus di Cesena bisogna solo memorizzare la relazione con il laboratorio, che a sua volta dovrà memorizzare la relazione con il sensore di movimenti e la luce.

Ogni DT dovrà essere in grado di mantenere una sua rappresentazione semantica che potrà comunicare al KGE quando richiesto.

La piattaforma *White Label Digital Twin Framework* (WLDT) permette la rappresentazione via software di un determinato *Physical Asset* (PA). Per mettere in collegamento il PA con il DT è necessario realizzare un *Physical Adapter* che permetterà di interfacciarsi con il PA, un *Digital Adapter* per interfacciarsi con il DT ed infine una funzione di *Shadowing* che metterà in comunicazione questi due componenti [16].

A questo proposito, si può procedere alla progettazione dei *Physical Adapter*. Il sensore di movimenti e la luce dovranno essere in grado di ricevere attraverso MQTT le informazioni dai relativi PA. Il protocollo MQTT è del

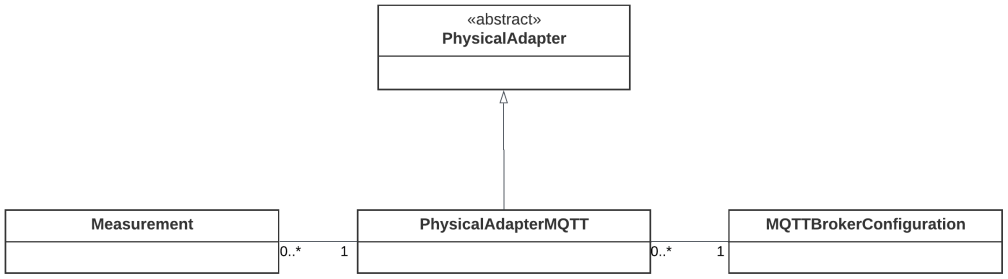

Figura 3.2: Diagramma delle classi UML per i Physical Adapter MQTT

tipo publish/subscribe, ovvero sono presenti due tipologie di entità (client), una che si occupa di pubblicare dei messaggi su un topic (simile all'idea di oggetto del messaggio) ed un'altra che si iscrive a quest'ultimo per leggerne il contenuto; il punto di contatto fra i due client è detto *MQTT broker*, che si occupa di ricevere i messaggi dal publisher, filtrarli ed inviarli ai client che si sono iscritti al topic del messaggio in oggetto [17]. Dato che nel caso preso in oggetto il broker sarà comune alle due entità, sarà possibile riunire le informazioni relative a quest'ultimo, come ad esempio indirizzo IP, porta, QoS o credenziali di accesso. Quindi, per quanto riguarda i *Physical Adapter* MQTT la struttura sarà quella proposta in figura 3.2.

I *Physical Adapter* del laboratorio e del campus di Cesena si occuperanno di pubblicare al loro avvio e memorizzare le relazioni che li legano agli altri DT.

A questo punto è necessario affrontare la realizzazione dei *Digital Adapter*, questi infatti rispetteranno la progettazione indicata in fase di progettazione, e l'obiettivo è mantenere una rappresentazione semantica aggiornata del DT. Per questo è stata realizzata una classe *Entity Knowledge Graph* Rapresentation che si occuperà di mantenere la rappresentazione del DT.

Le tecnologie per sviluppare knowledge graph sono svariate (es. GraphDB, Neo4j...) ma si è scelto di utilizzare Apache Jena, in quanto pienamente supportato ed open-sorce, caratteristica che lo rende il linea con il progetto WoDT e WLDT. La classe *Entity KG Rapresentation* mantiene al suo interno un modello di Apache Jena che viene aggiornato al cambiamento di una proprietà o alla creazione/rimozione di una instanza di relazione ed è progettata per ottenere, se richiesto, una stringa "RDF/XML" con la rappresentazione attuale.

Ad esempio di seguito c'è la rappresentazione ottenuta dal DT Light.

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:saref4bldg="https://saref.etsi.org/saref4bldg/"
    xmlns:saref="https://saref.etsi.org/core/">
  <rdf:Description rdf:about="http://localhost:3003/light">
    <saref:hasState>saref:OnState</saref:hasState>
    <rdf:type>https://saref.etsi.org/saref4bldg/Lamp</rdf:type>
  </rdf:Description>
</rdf:RDF>
```

Listato 3.1: Rappresentazione RDF/XML di Light

Per separare la scelta ontologica dalla realizzazione del DT, si è scelto di far uso di un'altra interfaccia, ovvero *Entity Ontology*, che risulta come una sorta di traduttore tra le proprietà e le relazioni del e quelle previste dall'ontologia. Ogni DT possiederà una implementazione di *Entity Ontology*, che mapperà le proprietà del DT con la relativa proprietà prevista dalla ontologia scelta.

3.3 Collegamento fra Digital Twin e WoDT Platform

A questo punto sono stati implementati dei DT funzionanti sfruttando la libreria WLDT, ma ora è necessario collegarli al relativo *WoDT Node*. É necessario aggiungere un modulo al DT, ovvero la *Management Interface*, che andrà a comunicare con il server HTTP REST dell'ecosistema. Il *Management Interface* infatti per prima cosa chiederà la registrazione del DT, per poi metterlo nello stato running, aggiornare lo stato del DT e creare/rimuovere relazioni con altri DT. Nel progetto esistente, il *Management Interface*, descrive lo stato del DT attraverso un *Digital Twin Descriptor*; questo strumento viene riutilizzato, estendo la possibilità di incapsulare lo stato RDF/XML del DT. Il *Digital Twin Descriptor*, nella progettazione originale prevede funzionalità molto più estese rispetto alla sola rappresentazione, ma la sua implementazione completa non verrà affrontata in questa tesi.

Il DT a questo punto può inviare tramite il client OkHttp3 il *Digital Twin Descriptor* formattato in JSON tramite il modulo GSON. Le relazioni invece vengono inviate come *Digital Twin State Relationship*, sempre formattate con GSON. Sebbene fino a questo punto l'implementazione del modulo per *WoDT platform* sia la stessa proposta nel progetto originale, si è deciso di aggiungere un altro componente, in quanto, per poter mantenere l'integrità dei dati, si è deciso di implementare un'autenticazione molto semplice lato DT; ovvero ogni

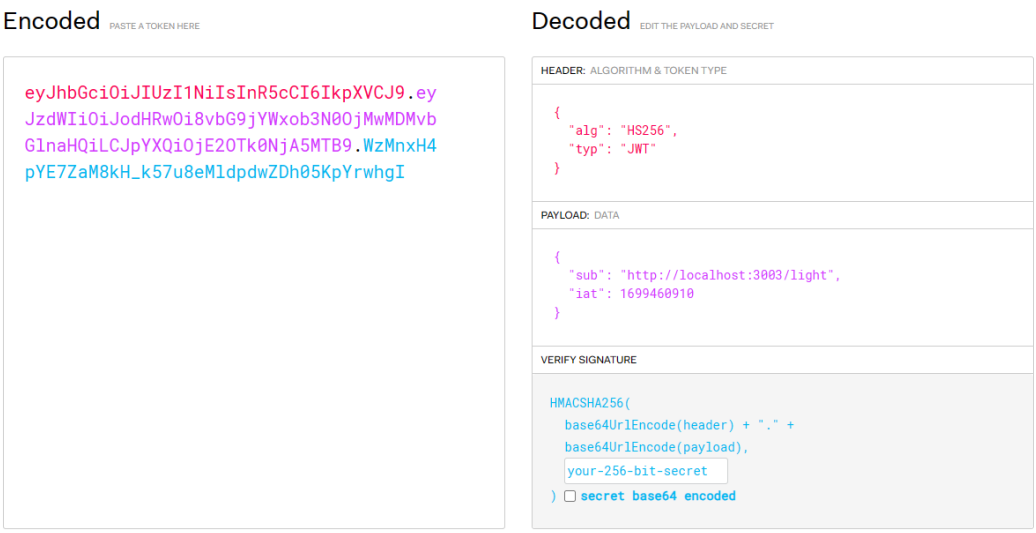

Figura 3.3: Esempio di token JWT

DT che richiede la registrazione nel *WoDT Platform*, se va a buon fine, riceve indietro un token che dovrà utilizzare per poter effettuare tutte le modifiche successive. Con questa modalità, solo chi ha il token può modificare una determinata risorsa del KGE. La modalità di autenticazione scelta è JSON Web Token (JWT), ovvero uno standard aperto (RFC 7519) che definisce un modo compatto per trasmettere in modo sicuro informazioni tra due parti attraverso un oggetto JSON [3], ed è stato scelto per implementare una forma di autenticazione molto semplice per questo progetto. Un token JWT è composto da tre parti separati da un punto, un *header* dove è indicato il tipo di token e l'algoritmo utilizzato, un *payload* dove sono presenti le informazioni che si vuole passare, nel nostro caso l'URI del DT ed il *timestamp* della creazione del token ed infine una firma che contiene anche una chiave segreta (un esempio di token è rappresentato in figura 3.3). In caso di necessità di maggiore sicurezza, si potranno implementare algoritmi di rotazione delle chiavi che non verranno affrontate in questa tesi. La creazione e la convalida dei token avvengono a livello del *Platform Internal Interface*.

3.4 Knowledge Graph Engine

Il cuore di questo progetto è rappresentato dal *Knowledge Graph Engine* (KGE), il quale è realizzato con una classe che implementa le tre interfacce specificate in fase di progettazione, queste interfacce prevedono dei metodi che ricalcano il *pattern observer* tipico della modellazione software ad oggetti.

La tecnologia utilizzata per memorizzare le triple RDF è sempre Apache Jena, scelta perché open-source e prevede anche un modulo per effettuare delle operazioni SPARQL chiamato ARQ.

Nel costruttore del KGE viene inizializzata una mappa che si occupa di associare ad ogni ID del DT il suo URI, inoltre vengono caricate le ontologie passate in input ed i relativi prefissi. Da notare che in Apache Jena esistono diverse tipologie di modelli su cui poter inserire le triple, nel nostro caso è importante creare un "OntModel", ovvero un modello capace di gestire classi, proprietà ed individui mantendendo la logica prevista dall'ontologia; se usassimo un "DefaultModel" ad esempio, perderemmo tutte le gerarchie previste dall'ontologia scelta, ripercuotendosi nell'efficacia di tutto il sistema.

Il primo passo prevede l'implementazione dei metodi previsti dal *Digital Twin Registration Listener*, ovvero dei metodi che vengono richiamati per la creazione e lo stop dei DT. Quando un DT viene registrato, viene inserito nella mappa che lega ID del DT ed il relativo URI e viene creata la risorsa nel modello di Jena; invece lo stop rimuove la risorsa nel *triple store*. Successivamente si passa ai metodi del *Digtal Twin Graph Listener* che si occupano dell'aggiornamento delle proprietà e delle istanze, ed infine si aggiunge il *Digital Twin Request Query Listener* per supportare le query.

3.4.1 Gestione aggiornamento delle proprietà

I DT, quando si aggiornano, inviano una loro rappresentazione RDF al *WoDT Platform* che si occuperà di recuperare la risorsa a cui si riferisce, caricare l'RDF ricevuto e confrontare l'RDF aggiornato con quello vecchio, eseguendo le modifiche necessarie. Quando si aggiorna una proprietà si controlla se il valore corrisponde ad un URI, e nel caso affermativo viene aggiunto come una risorsa; infatti quando Jena legge il valore di una proprietà RDF non è capace di distinguere un URI da un *literal* (ovvero un tipo primitivo), e di default viene letta come se appartenesse al secondo caso. Si è escelto di effettuare un controllo manuale, ovvero se il valore della proprietà inizia con "http" viene considerato URI, altrimenti viene aggiunto come un *literal*. Questo aspetto risulta molto importante per quanto riguarda le interrogazioni che non risulterebbero accurate, infatti si rischierebbe di perdere tutte le relazioni fra le risorse.

```
@Override
public void onDigitalTwinDescriptorUpdate(DigitalTwinURI
    digitalTwinURI,
    DigitalTwinDescriptorMessage digitalTwinDescriptor) {
```

```
        // Check if the digitalTwinURI and the URI in the descriptor
            are the same
        if (digitalTwinURI.getURI()
            .equals(this.idUriMap.get(digitalTwinDescriptor.getId())
            .getURI())) {
            Model receivedModel = ModelFactory.createDefaultModel();

            // Read the message in the RDF/XML base
            receivedModel.read(
                new java.io.StringReader(
                    digitalTwinDescriptor.getRapresentation()),
                "RDF/XML");

            updateProperties(receivedModel
                .getResource(digitalTwinURI.getURI()),
                    this.ontologyModel.getIndividual(
                        digitalTwinURI.getURI()));
        }
    }

    // Check if there are some properties to update
    private void updateProperties(Resource resourceUpdatedMessage,
            Resource resourceToUpdate) {
        resourceUpdatedMessage.listProperties()
            .toList()
            .stream()
            .forEach(stmt -> {
                Property predicate = stmt.getPredicate();

                if (resourceToUpdate.hasProperty(predicate))
                    resourceToUpdate.removeAll(predicate);

                // Check if is an URI or a Literal
                if (stmt.getObject().toString().startsWith("http"))
                    resourceToUpdate.addProperty(predicate,
                    this.ontologyModel.getOntClass(
                        stmt.getObject().toString()));
                else
                    resourceToUpdate.addProperty(predicate,
                        stmt.getObject());
            });
    }
```

Listato 3.2: Aggiornamento del descriptor nel Knowledge Graph Engine

3.4.2 Gestione delle relazioni

Quando viene creata un'istanza di una relazione fra DT è necessario per prima cosa ottenere l'individuo a cui andrà aggiunta, per poi aggiungere la proprità che avrà come valore l'URI di un'altra risorsa; questo è di particolare importanza in quanto, se fosse aggiunta come proprietà semplice, verrebbe considerata un *literal*, e quindi non verrebbe vista come relazione fra nodi, bensì come una semplice stringa. La rimozione delle istanze è molto intuitiva, si cerca la relazione in oggetto e la si cancella.

3.4.3 Gestione delle query

Il progetto *WoDT Platform* fin qui implementato non prevede un *observer* da poter richiamare dalla *Platform External Interface* per effettuare interrogazioni sul KGE; per questo è stata realizzata una interfaccia, così come accennato in fase di progettazione, da implementare nel KGE per poter effettuare delle query. La tecnologia utilizzata per implementare le interrogazioni su un'istanza di Apache Jena è ARQ, ovvero un motore per le interrogazioni che supporta SPARQL. Il risultato della query può essere formattato in diversi modi (es. XML, JSON, tabellare...) ma si è scelto JSON per una questione di compatibilità e flessibilità.

```
// Take in input a query, and return the result a JSON string
public String onDigitalTwinQueryRequest(String stringQuery) {
    Query query = QueryFactory.create(stringQuery);
    try {
        // Create and execute the query
        QueryExecution queryExecution =
            QueryExecutionFactory.create(query,
                this.ontologyModel);
        ResultSet resultSet = queryExecution.execSelect();

        // Return the result as a JSON string
        ByteArrayOutputStream outputStream = new
            ByteArrayOutputStream();
        ResultSetFormatter.outputAsJSON(outputStream, resultSet);

        return outputStream.toString();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

Listato 3.3: Esecuzione delle query nel Knowledge Graph Engine

3.5 API per le query

L'ultimo tassello riguarda l'implementazione del servizio REST lato applicativo per le interrogazioni sul grafo, che è la funzionalità più importante che offrirà questo progetto, in quanto, grazie a questa, sarà possibile navigare il *knowledge graph* sfruttando tutte le potenzialità che offre un linguaggio per le interrogazioni. Il linguaggio scelto è SPARQL, in quanto standard W3C e previsto dal modulo ARQ per Jena. Per implementare questa funzionalità è necessario aggiungere la possibilità nel *Platform External Interface* di ricevere dei messaggi HTTP passando come parametro GET la query ed eseguirla nel KGE.

Così come per l'aggiornamento dello stato dei DT, anche per le query esiste il rischio di compromettere l'integrità dei dati, in quanto da SPARQL 1.1 è nata la possibilità di modificare i dati del database attraverso l'uso degli operatori "INSERT", "MODIFY" e "DELETE". Per questo motivo è stato aggiunto un controllo sulla query prima della sua esecuzione, ovvero viene controllato se nella richiesta sia presente uno di questi operatori, restituiendo in caso affermativo al *client* un messaggio di "Operazione non autorizzata".

Capitolo 4

Validazione Progetto e Sviluppi Futuri

In questo capitolo verrà utilizzato lo scenario di riferimento modellato ed implementato nei capitoli precedenti per validare il progetto e mostrare degli esempi d'uso attraverso delle interrogazioni.

4.1 Esempi d'uso con query SPARQL

Sono proposti due esempi per verificare il corretto sviluppo del progetto, entrambi, oltre a verificare il corretto funzionamento del KGE, verificheranno se l'ontologia è stata caricata correttamente e viene sfruttata dal motore per le interrogazioni.

4.1.1 Stato dei dispositivi nel laboratorio 2.2

Questa interrogazione chiede al Knowledge Graph Engine quali sono i dispositivi che sono contenuti all'interno del laboratorio 2.2 ed il loro stato, oltre a verificare il funzionamento del triple store, ci permette di verificare il corretto funzionamento dell'ontologia scelta, ovvero Saref4bldg, in quanto ci aspettiamo di vedere l'entità "Light" che però è modellata come un entità Lamp, quindi il knowledge graph dovrà essere in grado di valutare correttamente anche le specializzazioni dell'entità "Device". Questa è la query SPARQL:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX saref: <https://saref.etsi.org/core/>
PREFIX saref4bldg: <https://saref.etsi.org/saref4bldg/>

SELECT ?device ?status
```

```
WHERE {
  ?device rdf:type saref:Device .
  <http://localhost:3002/lab22> saref4bldg:contains ?device .
 ?device saref:hasState ?status .
}
```

Listato 4.1: Query SPARQL per ottenere lo stato dei dispositivi presenti nel Lab 2.2

Il risultato in JSON ha restituito sia la variabile device che la variabile status: il device corrisponde correttamente all'entità "Light" modellata nello scenario di rifermineto e lo status corrisponde a saref:OnState, ovvero una classe dell'ontologia Saref che indica lo stato attivo di un determinato dispositivo. Lo stato corrisponde al valore comunicato tramite protocollo MQTT al DT simulando una reale lampadina.

```
{
  "head": {
    "vars": [
      "device",
      "status"
    ]
  },
  "results": {
    "bindings": [
      {
        "device": {
          "type": "uri",
          "value": "http://localhost:3003/light"
        },
        "status": {
          "type": "uri",
          "value": "https://saref.etsi.org/core/OnState"
        }
      }
    ]
  }
}
```

Listato 4.2: Risposta alla query in formato JSON

4.1.2 Misurazioni dei sensori nel laboratorio 2.2

La seconda query interroghera il *triple store* sui sensori contenuti nel laboratorio 2.2 ed il valore da loro misurato, compreso l'istante in cui è stata effettuata la misurazione.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX saref: <https://saref.etsi.org/core/>
PREFIX saref4bldg: <https://saref.etsi.org/saref4bldg/>

SELECT ?sensor ?timestamp ?value
WHERE {
   ?sensor rdf:type saref4bldg:Sensor .
   <http://localhost:3002/lab22> saref4bldg:contains ?sensor .
   ?sensor saref:makesMeasurement ?measurement .
   ?measurement saref:hasTimestamp ?timestamp .
   ?measurement saref:hasValue ?value .
}
```

Listato 4.3: Query SPARQL per ottenere il valore ed l'istante della misurazione effettuata dai sensori nel Lab 2.2

La risposta fornisce tutte e tre le variabili richieste, il sensore è rappresentato correttamente come un URI, il "timestamp" ed il "value" sono restituiti correttamente come literal, nello specifico, il "timestamp" è un valore dateTime e il "value" un valore booleano.


```
{
  "head": {
    "vars": [
      "sensor",
      "timestamp",
      "value"
    ]
  },
  "results": {
    "bindings": [
      {
        "sensor": {
          "type": "uri",
          "value": "http://localhost:3004/movementSensor"
        },
        "timestamp": {
          "type": "literal",
          "datatype": "http://www.w3.org/2001/XMLSchema#dateTime",
```

```
        "value": "2023-10-15T10:30:00Z"
      },
      "value": {
        "type": "literal",
        "datatype": "http://www.w3.org/2001/XMLSchema#boolean",
        "value": "true"
      }
    }
  ]
 }
}
```


Listato 4.4: Risposta alla query in formato JSON

4.2 Sviluppi futuri

Questo progetto di tesi aveva lo scopo di realizzare una prima implementazione di knowledge graph per il framework WoDT, in modo tale da poter testare le funzionalità di questo componente e svolgere il primo passo in direzione KG nel framework. Questo progetto potrà essere esteso sotto svariati punti di vista, come una gestione delle autenticazioni più complessa e sicura, realizzare una interfaccia in grado di poter mostrare il grafo ed interagire con esso oppure fornire una API lato applicativo che implementi solo un numero limitato di funzioni da esporre piuttosto che le funzionalità dell'intero linguaggio SPARQL (per questioni di sicurezza).

Per come il WoDT è stato pensato inoltre, il progetto convergerà sempre di più verso una struttura distribuita e quindi il seguito naturale di questo progetto è la realizzazione di un Knowledge Graph distribuito[15].

Conclusioni

Questo progetto di tesi è nato con l'obiettivo di realizzare il componente Knowledge Graph Engine per la piattaforma Web of Digital Twin, che a sua volta si pone l'obiettivo di creare un ecosistema open-source di DT, spostando quindi lo sguardo dalla entità singola alla rete di entità interconnesse.

È stata affrontata la gestione della semantica da parte dei DT, aggiungendo i componenti necessari quali l'*Entity KG Rapresentation* e l'*Entity Ontology Interface* che hanno reso possibile la traduzione dello stato interno del DT nella corrispettiva rappresentazione semantica. Inoltre, il progetto realizzato ha aggiunto una implementazione molto semplice, all'interno del WoDT Platform, del KGE, che permette di memorizzare le informazioni dei vari DT attraverso una rappresentazione semantica, implementata a livello del DT. É stato affrontato, seppure in modo superficiale, il tema della sicurezza: l'integrità dei dati può essere messa a rischio se non si effettua nessun tipo di controllo su chi modifica le informazioni in uno storage centralizzato. Infine, è stata estesa la API REST a livello applicativo per poter effettuare delle query SPARQL sul grafo, questo è lo strumento che, oltre a valutare l'efficacia del lavoro svolto, permette di estendere a livello applicativo le funzionalità sfruttabili della piattaforma.

Questo progetto rappresenta un contributo, seppur non completo in tutti i suoi aspetti, all'implementazione dei knowledge graph nella piattaforma WoDT, che potrà essere poi estesa con la tecnologia relativa ai knowledge graph distribuiti.

Ringraziamenti

Un ringraziamento al Prof. Alessandro Ricci, al Prof. Marco Picone e il Dott. Samuele Burattini per avermi proposto e guidato in questa tesi.

Un grazie speciale ai miei genitori per avermi sempre supportato in questo percorso ed avere sempre creduto in me.

Ringrazio molto Beatrice per avermi sostenuto ed essere stata al mio fianco nell'ultimo anno e Giulia per riuscire a strapparmi un sorriso anche nei momenti difficili.

Infine ringrazio anche il mio amico Luca per le numerose colazioni e chiacchierate passate negli ultimi anni.

Bibliografia

[1] Grigoris Antoniou and Frank van Harmelen. Web ontology language: Owl. *Handbook on ontologies*, pages 91–110, 2009.

[2] Grigoris Antoniou and Frank Van Harmelen. *A semantic web primer*. MIT press, 2004.

[3] Auth0. Introduction to json web tokens. `https://jwt.io/introduction`. Online; accessed 8 November 2023.

[4] Barbara Rita Barricelli, Elena Casiraghi, and Daniela Fogli. A survey on digital twin: Definitions, characteristics, applications, and design implications. *IEEE access*, 7:167653–167671, 2019.

[5] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific american*, 284(5):34–43, 2001.

[6] Min Deng, Carol C Menassa, and Vineet R Kamat. From bim to digital twins: A systematic review of the evolution of intelligent building representations in the aec-fm industry. *Journal of Information Technology in Construction*, 26, 2021.

[7] Bob DuCharme. *Learning SPARQL: querying and updating with SPARQL 1.1.* " O'Reilly Media, Inc.", 2013.

[8] ETSI. Saref extension for building. `https://saref.etsi.org/saref4bldg/v1.1.2/`, 2020. Online; accessed 20 October 2023.

[9] ETSI. Saref: the smart applications reference ontology. `https://saref.etsi.org/core/v3.1.1/`, 2020. Online; accessed 20 October 2023.

[10] Pascal Hitzler. A review of the semantic web field. *Communications of the ACM*, 64(2):76–83, 2021.

[11] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F Patel-Schneider, Sebastian Rudolph, et al. Owl 2 web ontology language primer. *W3C recommendation*, 27(1):123, 2009.

[12] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. Knowledge graphs. *ACM Computing Surveys (Csur)*, 54(4):1–37, 2021.

[13] Eric Miller. An introduction to the resource description framework. *Bulletin of the American Society for Information Science and Technology*, 25(1):15–19, 1998.

[14] Renato Iannella, Semantic Identity, James McKinney and OpenNorth. vcard ontology - for describing people and organizations. `https://www.w3.org/TR/vcard-rdf/`, 2014. Online; accessed 04 October 2023.

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

[16] Marco Picone Samuele Burattini. Whitelabel digital twin framework. `https://github.com/wldt/wldt-core-java`, 2023. Online; accessed 23 October 2023.

[17] Dipa Soni and Ashwin Makwana. A survey on mqtt: a protocol of internet of things (iot). In *International conference on telecommunication, power analysis and computing techniques (ICTPACT-2017)*, volume 20, pages 173–177, 2017.

[18] Marta Spadoni. Progettazione e sviluppo di una piattaforma per ecosistemi di digital twin. Tesi di laurea magistrale, Alma Mater Studiorum Università di Bologna, 2021-2022. Relatore: Alessandro Ricci, Correlatori: Marco Picone, Samuele Burattini.

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

[20] Jihong Yan, Chengyu Wang, Wenliang Cheng, Ming Gao, and Aoying Zhou. A retrospective of knowledge graphs. *Frontiers of Computer Science*, 12:55–74, 2018.