UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Elaborato finale in **Intelligenza Artificiale**

Utilizzo di FIWARE come piattaforma per le Smart Solutions: architettura e metodologie

Anno Accademico 2023/2024

Candidato:

Raffaele Mele

matr. N46003834

Dedicato a chi mi ha insegnato ad

affrontare con coraggio e

determinazione gli ostacoli della vita

*…**alla mia mamma e al mio papà.***

Indice

Introduzione

Il progredire della tecnologia digitale sta modificando radicalmente la vita delle persone e così il concetto stesso di *bene*. In questa nuova realtà, il bene più prezioso sono le *informazioni*, la cui raccolta, analisi e gestione in modo efficace ed efficiente risulta essere ormai indispensabile. L'Unione Europea (UE) ha messo in atto una propria strategia al fine di prepararsi per la nuova era digitale con l'obiettivo di porre la tecnologia al servizio dei cittadini e delle imprese. Pertanto, l'UE ha deciso di perseguire una strada differente da quella di altri Paesi, al fine di rafforzare la propria sovranità digitale e rivolgendo maggiore interesse ai *dati*, alla *tecnologia* e alle *infrastrutture*. Inoltre, la presidente dell'Unione Europa, Ursula von der Leyen, ha più volte sottolineato che uno degli altri obiettivi primari dell'UE è garantire uno sviluppo antropocentrico, trasparente e responsabile dell'*Intelligenza Artificiale.* A tale scopo, è stato stilato un programma strategico per il decennio digitale che fissa i traguardi e gli obiettivi per il 2030 per guidare, gradualmente, la trasformazione digitale dell'Europa. I punti cardine di questo programma sono quattro:

- **Competenze:** almeno l'80% della popolazione adulta dovrebbe possedere competenze digitali di base e 20 milioni di specialisti dovrebbero essere impiegati nell'UE nel settore delle tecnologie dell'informazione e della comunicazione, a parità di genere.

- **Trasformazione digitale delle imprese:** circa il 75% delle imprese dovrebbe utilizzare servizi di cloud computing, big data e intelligenza artificiale; oltre il 90% delle

PMI dovrebbe arrivare ad un livello base di competenze digitali e, inoltre, dovrebbe raddoppiare il numero delle imprese "unicorno" nell' UE.

- **Infrastrutture digitali sicure e sostenibili:** Si è stimato che tutte le famiglie dell'UE dovrebbero beneficiare di una connettività Gigabit e tutte le zone abitate dovrebbero essere coperte dal 5G. La produzione di semiconduttori sostenibili e all'avanguardia in Europa dovrebbe rappresentare il 20% della produzione mondiale. Inoltre, si prevede l'installazione di 10 000 nodi periferici a impatto climatico zero e altamente sicuri e la creazione del primo computer quantistico europeo.

- **Digitalizzazione dei servizi pubblici:** Tutti i servizi pubblici fondamentali dovrebbero essere disponibili online, tutti i cittadini avranno quindi accesso alla propria cartella clinica elettronica e l'80% della popolazione dovrebbe utilizzare l'identificazione digitale (eID).

Si evidenzia chiaramente che la trasformazione digitale interessa ogni aspetto della vita dei cittadini e l'UE, al fine di concretizzare quanto pianificato, ha stilato la *"Dichiarazione europea sui diritti e i principi digitali".* Tale dichiarazione ribadisce la figura del cittadino come attore principale di tale trasformazione e l'UE come guida di responsabili politici e imprese che si occupano delle nuove tecnologie. Pertanto, il *software* che sarà necessario per guidare la trasformazione digitale dovrà allontanarsi dal concetto di "software proprietario" ma essere invece aperto o, per utilizzare un termine ormai noto anche a non addetti ai lavori, "*open source*" [2]. Questo rappresenta il primo passo verso quella che può essere chiamata "*Smart Society*" o società intelligente. Quest'ultima risulta essere basata proprio sugli obiettivi che l'UE si è posta di raggiungere entro il 2030. In particolare, sono tre i pilastri della Smart Society ovvero: la ***connettività***, i ***dispositivi intelligenti*** o "dispositivi IoT" e lo ***sviluppo software***. Il primo pilastro comprende reti legacy (wireless, mobili, fisse, satellitari e via cavo) e nuove tecnologie IoT recenti; giocano, quindi, un ruolo chiave la connettività e l'accesso affidabile ai dati in

tempo reale. I dispositivi IoT, invece, sono oggetti connessi che generano dati ed eseguono attività da remoto. Infine, lo sviluppo software rappresenta l'elemento chiave di collegamento fra i dispositivi e la rete. In questo modo, grazie alla cooperazione di due o più dispositivi tra loro, si è in grado di creare nuovi servizi e applicazioni (*interoperabilità)*. Queste applicazioni tendono sempre più ad essere gestite in cloud, connesse all' *Internet of Things* e capaci di analizzare informazioni su larga scala. Al fine di facilitare la transizione verso soluzioni innovative (***Smart Solutions***) in questa nuova, complessa e rapida realtà digitale, entra in gioco il framework ***FIWARE.*** Quest'ultimo è una piattaforma cloud open source che, attraverso nuove tecnologie, *API* (Application Programming Interface) già implementate e la definizione di standard, permette di costruire e gestire applicativi e servizi all'avanguardia rendendo i complessi processi di sviluppo più agili, abbattendo i costi di produzione, garantendo al tempo stesso alta qualità e, infine, preservando sicurezza e privacy. Ad oggi, tutti i "tools" di FIWARE sono disponibili e pronti all'uso in un proprio ambiente aperto noto come "***FiwareLab***", dove sviluppatori e imprenditori digitali hanno gli strumenti necessari per dare vita ai propri progetti e testare le applicazioni con dati reali. Pertanto, viene facilitato lo sviluppo di applicazioni e servizi per: *Smart Cities, eHealth, eLearning, Smart Energy, Smart Agri-food* e *Smart Industry*, dove la qualità e la quantità delle informazioni in gioco è rilevante. FIWARE, come definito dai fondatori, risulta essere un vero e proprio ***ecosistema*** costituito di data centers propri e nodi diffusi nei diversi Paesi. Viene, inoltre, supportato da una larga community dove imprenditori, pubbliche amministrazioni ed esperti del settore incontrano idee innovative da sostenere e supportare sia finanziariamente che tecnicamente. Il presente lavoro di tesi si pone l'obiettivo di fornire un quadro generale sulla piattaforma FIWARE, su come è stata concepita e il suo funzionamento. Inoltre, verranno illustrati due casi d'uso di FIWARE nell'ambito dei *Digital Twin*. Gli approfondimenti proposti sono derivanti anche da un'esperienza condotta presso il "*Competence Center MedITech*" di Napoli. Il Mediterranean Competence Centre 4 Innovation (*MedITech 4.0*) è il Centro di Competenza poli-regionale, attivo in Puglia e Campania, nato come facilitatore nell'adozione delle tecnologie abilitanti dell'Industria

4.0 da parte delle PMI (Pubbliche e Medie Imprese) e della PA (Pubblica Amministrazione), grandi protagonisti della transizione digitale. MedITech, finanziato dal MISE (Ministero delle Imprese e del Made in Italy) e dall'UE, si pone l'obiettivo di coniugare competenze tecnologiche di eccellenza con le potenzialità di innovazione dei sistemi produttivi e socioeconomici. Da tempo MedITech è membro della Fondazione FIWARE e rappresentante nel Mezzogiorno delle potenzialità della piattaforma europea open source. Presso il centro MedITech di Napoli, infatti, è presente un nodo di FIWARE chiamato *NAPOLI1*, specializzato in Digital Twin e Data Space, il quale beneficia di un cluster di server per incoraggiare e facilitare la sperimentazione della piattaforma FIWARE.

Capitolo 1: FIWARE

La fondazione FIWARE è un'organizzazione no-profit, fondata nel 2016, che guida la definizione e incoraggia l'adozione di standard aperti, implementati attraverso tecnologie open-source che facilitano lo sviluppo di soluzioni intelligenti. Il motto della fondazione è, infatti, "*Open APIs for Open Minds"*, che riflette precisamente la *mission* di FIWARE. In Figura [Fig.1], viene riportato il logo ufficiale delle Fondazione.

Figura 1: Logo di FIWARE

"*Open APIs"* evidenzia la volontà di voler definire un set di standard universali per la gestione dei *context data* al fine di facilitare gli sviluppatori nella creazione di applicazioni. Invece, "*Open Minds*" si rivolge agli investitori e agli sponsor poiché, per affermarsi con decisione nel nuovo mercato digitale, è necessario collaborare e confrontarsi con realtà spesso diverse tra loro. Per lo sviluppo delle *Smart Solutions*, è necessario raccogliere informazioni da diverse sorgenti, che non si limitano ai soli dispositivi IoT. Quest'ultime sono poi processate e analizzate per implementare il comportamento intelligente desiderato. In questo scenario, i dati d'ambiente o "*context data*" assumono un ruolo fondamentale, tanto è che si è soliti utilizzare l'espressione: "*context matters*" ovvero il contesto conta. Tutti questi concetti verranno, comunque, di seguito approfonditi nel dettaglio, al fine di proporre una visione complessiva dell'ecosistema FIWARE.

1.1 Architettura FIWARE

Dal punto di vista tecnico, FIWARE è costituito da elementi chiamati *Generic Enablers (GE)*, a loro volta, costituiti da un gruppo di componenti che forniscono un set di funzionalità rese disponibili tramite API e interfacce tra loro interoperabili. FIWARE mette a disposizione un catalogo di Generic Enablers (FIWARE Catalogue), dove, per ogni GE, è presente l'interfaccia e l'implementazione di riferimento. Il componente principale e obbligatorio di un'architettura FIWARE è il "***Context Broker Generic Enabler***" e deve essere presente in qualsiasi soluzione intelligente affinché questa possa essere definita come "***Powered by FIWARE***". Infatti, il Context Broker rappresenta il punto di passaggio delle informazioni e funziona da intermediario tra le diverse componenti che possono essere aggiunte ad esso. Le informazioni sul contesto sono rappresentate attraverso valori assegnati agli attributi che caratterizzano le entità rilevanti per uno specifico contesto. Una delle caratteristiche più importanti del Context Broker GE è quella che consente di accedere alle informazioni di contesto in modo del tutto indipendente dalla sorgente delle stesse.

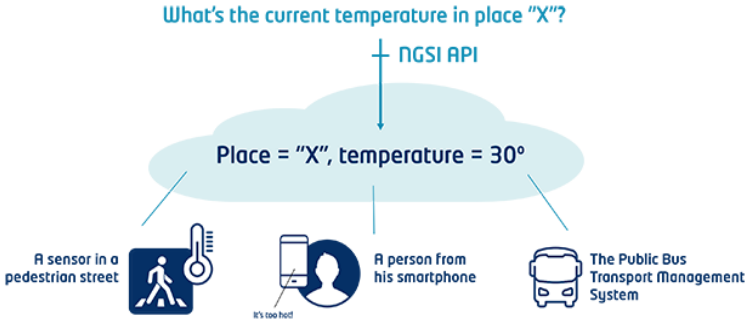

Figura 2: Schematizzazione di una generica Smart Solution

Per esempio, una generica applicazione potrebbe aver bisogno di conoscere quale sia la "Temperatura" in un certo "Luogo", indentificato da coordinate [Fig.2]. La temperatura può essere ottenuta tramite sensori installati, ad esempio, sui semafori presenti su una strada di quel luogo, oppure può essere rilevata attraverso sensori installati su autobus che circolano su una strada differente dalla precedente. Un modo del tutto analogo per ricevere tale informazione potrebbe essere quello di analizzare le segnalazioni fatte da utenti tramite i loro smartphone. Di conseguenza, il modo in cui tale applicazione ottiene il

valore di “Temperatura”, attraverso il Context Broker, è del tutto indipendente dalla natura della sorgente.

1.2 FIWARE NGSI API

Lo standard utilizzato da un FIWARE Context Broker è FIWARE NGSI v2 (Next Generation Service Interface). Tale API ha lo scopo di gestire l’intero ciclo di vita delle risorse di contesto tramite aggiornamenti, queries e il sistema notifications/subscriptions. Le specifiche del FIWARE NGSI si sono evolute con il tempo, fino ad allinearsi con lo standard ETSI NGSI-LD (European Telecommunications Standards Institute, organismo internazionale, senza scopo di lucro e indipendente, ufficialmente responsabile per la definizione ed emissione di standard in Europa nel campo delle tecnologie dell’informazione). L’API NGSI di FIWARE permette di definire:

- Un data model per le risorse di contesto, tramite il concetto di “context entities” ovvero le entità che costituiscono il contesto analizzato.
- Un’interfaccia per i context data per scambiare informazioni tramite query, subscription e operazioni di update.

1.2.1 Terminologia NGSI

In un data model NGSI, gli elementi fondamentali sono le context entities. Queste sono caratterizzate da “attributes” che, a loro volta, possiedono dei “metadata” [Fig.3].

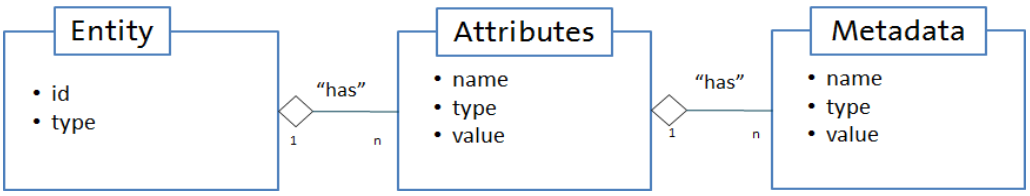

Figura 3: Entità, Attributi e Metadati NGSI

Le context entities, o entità, rappresentano un oggetto fisico o logico del mondo reale (per esempio: sensori, persone, una stanza... etc) e ogni entità è caratterizzata da un proprio campo, chiamato “entity id”. Dipendentemente dallo specifico FIWARE NGSI, le entità possono avere anche un altro campo, chiamato “entity type”. Tale campo ha funzione semantica, ovvero descrive il tipo di oggetto che viene rappresentato dall’entità (per

esempio: una context entity con id: “sensor-365”, potrebbe avere come campo tipo: “sensoreTemperatura”). Ogni entità è univocamente identificata dalla combinazione dei campi “id” e “type”. Invece, i context attributes sono le proprietà delle context entities; ad esempio, la velocità corrente di un aereo può essere modellata con l’attributo *current_speed* dell’entità *airplane-120.* Nel data model NGSI, un attributo è caratterizzato da:

- *attribute name*, nome dell’attributo: indica il tipo di proprietà che descrive l’attributo (nell’esempio precedente, current_speed è un valido attribute name).
- *attribute type*, tipo dell’attributo: indica a quale tipo di valore NGSI l’attributo si riferisce.
- *attribute value,* valore dell’attributo: contiene il valore effettivo o, alternativamente, metadati che descrivono le proprietà dell’attributo.
- *Metadata:* si trovano di frequente nel modello NGSI. Si utilizzano, in alternativa, come parte dell’attribute value, come sopra descritto. Ogni metadato è, a sua volta, costituto da: name, type e value.

1.2.2 Notazione JSON

Prima di procedere alla sintassi di un’entity nel modello NGSI, viene illustrato brevemente il formato adottato per rappresentarle, ovvero JSON (JavaScript Object Notation), un formato per lo scambio di dati. Tale standard risulta essere facile da leggere e scrivere per le persone ma, allo stesso tempo, risulta essere facilmente generabile e analizzabile per una qualsiasi macchina. Si basa su un sottoinsieme del linguaggio di programmazione JavaScript ed è diventato uno standard ECMA (European Computer Manufacturers Association, associazione dedita alla standardizzazione nel settore informatico e delle comunicazioni) nel 2013. JSON è un formato di testo completamente indipendente dal linguaggio di programmazione, ma utilizza una serie di convenzioni accettate da linguaggi come: C, C++, C#, Java e JavaScript, Python e molti altri. Ed è proprio per questo motivo che JSON è un formato ideale per lo scambio di dati. Il formato JSON è basato su due strutture:

- Un insieme di coppie nome/valore. Per fare un paragone con altri linguaggi di programmazione, ciò è simile a strutture come record, oggetto o struct.
- Un elenco ordinato di valori, similmente ad array o vettori.

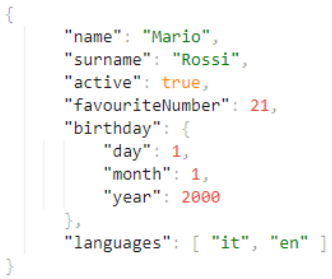

Figura 4: Esempio di oggetto JSON

Per creare un oggetto JSON è necessario [Fig.4]:

- Disporre una coppia di parentesi graffe, ad inizio e fine struttura, che conterranno l'intero oggetto.
- Inserire all'interno delle parentesi graffe una sequenza di proprietà, i cui campi "name" e "value" sono separati dal simbolo ":". Nell'esempio in figura, "name": "Mario" significa che l'oggetto possiede la proprietà "nome" il cui valore è impostato a "Mario".
- Tutte le proprietà dell'oggetto sono separate da una virgola. Nell'esempio in figura, si noti che la proprietà "birthday" risulta essere, a sua volta, un oggetto JSON, con le sue proprietà. Infine, "languages" è un array JSON, i cui valori sono ordinati e separati da virgola e racchiusi tra parentesi quadre ([valore1, valore2, … valore N]).

Si noti che, ai fini di questo lavoro di tesi, viene qui introdotto anche un'evoluzione del formato JSON ovvero JSON-LD, di cui si discuterà nei paragrafi a seguire.

1.2.3 Rappresentazione JSON di entità e attributi

Una entity è rappresentata da un oggetto JSON con la sintassi nella Figura [Fig.5].

```
{
  "id": "entityID",
  "type": "entityType",
  "attr_1": <val_1>,
  "attr_2": <val_2>,
  ...
  "attr_N": <val_N>
}
```

Figura 5: entità JSON

In particolare:

- Il campo “entityID” dell’entità è specificato dalla proprietà “id” dell’oggetto, il cui valore è una stringa contenente l’identificativo dell’entità.
- Il campo “entityType” è specificato dalla proprietà “type” dell’oggetto, il cui valore è, una stringa contenente il nome del tipo dell’entità.
- I campi <val_1> … <val_N> sono gli attributi dell’entità, la cui rappresentazione segue, anche in questo caso, il formato JSON. L’unica restrizione sui nomi degli attributi è quella di utilizzare nome diversi da “id” e “type”.

Un attributo, invece, è rappresentato da un oggetto JSON con la seguente sintassi [Fig. 6]:

```
{
  "value": <...>,
  "type": <...>,
  "metadata": <...>
}
```

Figura 6: attributo in formato JSON

Per i campi “value”, “type” e “metadata”, valgono le stesse considerazioni fatte precedentemente. Tuttavia, ci sono due modi di rappresentare le entità e questi devono essere supportati da ogni implementazione NGSI: keyValues mode e value mode, il cui vantaggio sta nel fatto che questi permettono di generare rappresentazioni semplificate di entità. In particolare:

- KeyValues mode: in questo modo si rappresentano gli attributi di un’entità solo con i loro valori, trascurando i campi “type” e “metadata” [Fig.7].

Figura 7: esempio di rappresentazione KeyValues

- Value mode: si rappresenta l'entità come un array di attributi. In questo caso, le informazioni riguardanti i campi "id" e "type" vengono tralasciate. Una variante di questa modalità è la "unique mode", dove i valori non vengono mai ripetuti [Fig.8].

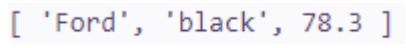

Figura 8: esempio di rappresentazione Value mode

1.2.4 Linked Data ed evoluzione degli standard JSON-LD e NGSI-LD

Come illustrato precedentemente, l'interfaccia NGSI v2 viene utilizzata per la creazione e manipolazione delle context entities e JSON come formato per rappresentarle. Un'evoluzione di quell'interfaccia ha creato una specifica supplementare, chiamata NGSI-LD (Linked Data), migliorando le context entities tramite il concetto di Linked Data. Tutti gli utenti del Web hanno, ormai, familiarità con il concetto di link (hypertext links), come questo sia capace di guidare un browser da una pagina web all'altra, tramite protocolli ben definiti. La creazione di un sistema di link richiede l'utilizzo di un formato per rappresentare i dati, cioè JSON-LD e l'assegnazione di identificativi unici (URLs o URNs) sia per le entità che per le relazioni tra esse. Le entità, collegate in questo modo, posso essere utilizzate per rispondere a domande relative al mondo dei Big Data e le relazioni possono essere "percorse" al fine di ottenere informazioni utili. JSON-LD risulta essere un metodo per evitare ambiguità quando sussistono relazioni tra entità, che sono rappresentate in formato JSON, e garantisce che tali Linked Data siano analizzabili dalle macchine. JSON-LD assicura, quindi, che tutti gli attributi dei dati possano essere facilmente confrontati quando questi provengono da sorgenti diverse. Per esempio, se due entità hanno un attributo "name", come può una macchina stabilire con sicurezza se quell'attributo si riferisce al "nome" di un oggetto piuttosto che al "nome" di un utente o luogo? Vengono quindi utilizzati URL e Data Model per rimuovere tale ambiguità,

consentendo agli attributi di avere sia una forma “breve” (come nel caso di “name”) che una più dettagliata (per esempio: http://schema.org/name). In questo modo, risulta semplice capire la semantica degli attributi in una struttura dati. In aggiunta, JSON-LD introduce il concetto di *“@context”*, elemento che fornisce informazioni ulteriori alla macchina al fine di interpretare con chiarezza e profondità il resto dei dati [Fig.9]. Tecnicamente, “@context” è un attributo che contiene puntatori a informazioni di contesto e consente di accedere a tutti i Data Models definiti dalla Fondazione FIWARE.

```
{
  "@context": "https://json-ld.org/contexts/person.jsonld",
  "@id": "http://dbpedia.org/resource/John_Lennon",
  "name": "John Lennon",
  "born": "1940-10-09",
  "spouse": "http://dbpedia.org/resource/Cynthia_Lennon"
}
```

Figura 9: esempio di entità con aggiunta del campo @context

Quindi, sfruttando le caratteristiche di JSON-LD, lo standard NGSI-LD permette a tali entità o dati di diventare “Linked” ovvero collegati. Il modello di dati NGSI-LD risulta essere più complesso di NGSI, con definizioni più rigide al fine di creare un grafo della conoscenza navigabile. Un grafo della conoscenza orientato, come quello in Figura [Fig.10], è costituto da un insieme di nodi e archi. I nodi rappresentano le entità NGSI e gli archi una relazione tra gli attributi tra due entità. Le entities rappresentano, come in NGSI, l’elemento centrale, ognuna deve possedere un “*id”* URN (Uniform Resource Name) unico, ovvero una sequenza di caratteri che indentifica universalmente ed univocamente una risorsa e, opzionalmente, un campo “*type*” usato per definire qual è la struttura dati che viene rappresentata.

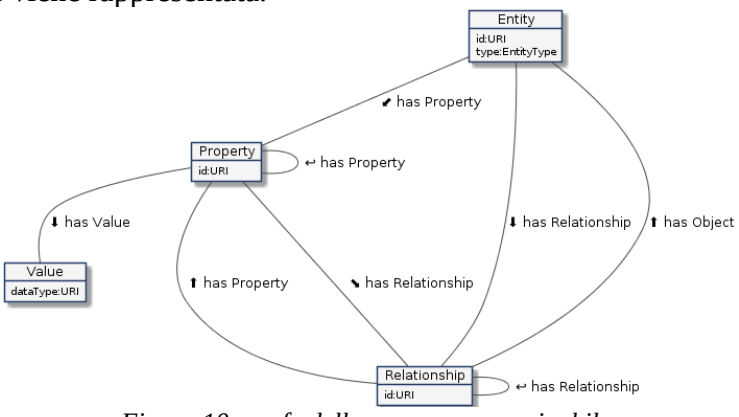

Figura 10: grafo della conoscenza navigabile

In questo modo, un'applicazione parte da un pezzo di Linked Data, segue i link, ed arriva ad altri frammenti di LD che sono presenti in altri siti del Web. Quindi risulta fondamentale, per il modello informativo NGSI-LD, la corretta creazione di LD "machine-readable".

1.3 Entità e Data Models

La Fondazione FIWARE si vuole porre da guida verso l'adozione di Data Models comuni e compatibili tra loro, che sostengano il mercato digitale di soluzioni intelligenti, interoperabili e replicabili in più settori. Generalmente, un Data Model comprende quattro elementi:

- Lo ***schema***, o una rappresentazione tecnica del modello che definisce la struttura e tipi dei dati.
- La ***specifica***, cioè una documentazione per chi utilizza il Data Model.
- Un ***URI*** (Uniform Resource Identifier) con URL contenente informazioni circa le entità e i relativi attributi.
- Una ***serie di esempi***, forniti per le versioni NGSI v2 e NGSI-LD.

Tutti i modelli di dati sono pubblici ed esenti da royalty. Circa quest'ultima modalità di licenza, sono garantiti tre diritti agli utenti quali: utilizzo, modifica e condivisione gratuita. I Data Models sono raggruppati per aree appartenenti ad uno o più settori industriali, e ogni area è un repository GIT. Il ciclo di sviluppo di un modello di dati segue il "*Manifesto for Agile Standardization*", nel quale si ribadiscono alcuni concetti chiave come: la sostenibilità, l'essere open-source, semplicità piuttosto che complessità e riutilizzo di ciò che è già a disposizione, ma soprattutto automatizzazione dei meccanismi al fine di ridurre i tempi di sviluppo delle soluzioni intelligenti [2]. Di conseguenza, gli utenti possono sviluppare nuovi Data Model, che tuttavia dovranno essere esaminati per renderli ufficiali. In questo stato, i Data Model si definiscono "***Incubated***". Una volta accettati, passano allo stato "***Harmonization***", che consiste nel renderli compatibili con altri modelli di dati o eventualmente modificarli. Infine, un Data Model completamente documentato, con uno schema ed esempi, passa allo stato di "***Official***". In Figura [Fig.11],

viene mostrato un esempio di Data Model per NGSI-LD relativo ad una batteria, insieme alle sue specifiche hardware.


```
{
  "id": "urn:ngsi-ld:Battery:santander:d95372df39",
  "type": "Battery",
  "acPowerInput": 1.5,
  "acPowerOutput": 0.5,
  "autonomyTime": "PT1H",
  "cycleLife": 20000,
  "dataProvider": "bike-in.com",
  "location": {
    "type": "Point",
    "coordinates": [-4.754444444, 41.640833333]
  },
  "rechargeTime": "PT6H",
  "refDevice": "urn:ngsi-ld:Device:santander:d95372df39",
  "source": "bike-in.com",
  "status": [
    "working"
  ],
  "@context": [
    "https://raw.githubusercontent.com/smart-data-models/dataModel.Battery/master/context.jsonld"
  ]
}
```


Figura 11: esempio di Data Model relativo ad una batteria

L'esempio in Figura è tratto dal repository GitHub relativo ai Data Models (github.com/smart-data-models), in particolare nel dominio Smart Energy, nel subject "dataModel.Battery". Offrendo una vasta gamma di Data Model così organizzati, si crea uno "***standard de-facto***".

1.3.1 Meccanismo Subscriptions e Notifications

Nell'ecosistema FIWARE, un'entità rappresenta un oggetto fisico o logico del mondo reale, caratterizzata da un proprio stato che varia nel tempo. Tale stato è in continua evoluzione dato il continuo flusso di dati provenienti, per esempio, da sensori IoT. Quindi, un'entità dovrà "reagire" ai cambiamenti del mondo reale. Pertanto, il sistema dovrà attivare un meccanismo per "informare" la "corretta" entità, nel momento in cui si verifica un evento che provochi una modifica dello stato (rappresentato dagli attributi dell'entità). Tale meccanismo è noto con il nome di "*Subscriptions/Notifications*", con il quale, un'applicazione può effettuare una "sottoscrizione" al fine di essere "informata" nel momento in cui avvengono cambiamenti. Il Context Broker Orion offre tale meccanismo di notifica asincrona in modo che le applicazioni possono mantenersi aggiornate rispetto a cambiamenti delle *context information.* In particolare:

- Una *Subscription* o sottoscrizione, ridurrà il numero di richieste di update e il traffico dati. Il sistema non dovrà quindi restare in una condizione di attesa attiva di

cambiamenti, ovvero di *polling*, e nemmeno effettuare delle *query* continuamente. Il vantaggio di ridurre il traffico dati in questo modo, sarà quello di migliorare drasticamente la reattività complessiva del sistema.

- Una *Notification* o notifica, si attiva dopo l'attivazione di una *Subscription* e "informa" le entità coinvolte nell'operazione di aggiornamento.

Di seguito, tramite due Figure con relativi esempi, vengono illustrati due modi per creare un meccanismo *subscription*/*notification*.

```
curl -iX POST \
  --url 'http://localhost:1026/v2/subscriptions' \
  --header 'content-type: application/json' \
  --data '{
  "description": "Notify me of all product price changes",
  "subject": {
    "entities": [{"idPattern": ".*", "type": "Product"}],
    "condition": {
      "attrs": [ "price" ]
    }
  },
  "notification": {
    "http": {
      "url": "http://tutorial:3000/subscription/price-change"
    }
  }
}'
```

Figura 12: esempio di subscription-notification

In Figura [Fig.12], viene illustrato il meccanismo in questione. Una *subscription* si effettua tramite una richiesta *POST* e il corpo di tale richiesta consiste di due parti: *subject* e *notification.* La sezione *subject*, che comprende i campi *entities* e *condition*, informa che la subscription (riferendoci all'esempio in Figura [Fig. 12]) verrà attivata ogni volta che l'attributo "***price"*** di qualsiasi entità "***Product"*** viene modificato. La sezione *notification*, invece, informa che, quando le condizioni della subscription sono verificate, viene eseguita una richiesta POST, contenente le entities "***Product***", all'URL specificato dall'omonimo campo "*url*".

```
curl -iX POST \
  --url 'http://localhost:1026/v2/subscriptions' \
  --header 'Content-Type: application/json' \
  --data '{
  "description": "Notify me of low stock in Store 001",
  "subject": {
    "entities": [{"idPattern": ".*","type": "InventoryItem"}],
    "condition": {
      "attrs": ["shelfCount"],
      "expression": {"q": "shelfCount<10;refStore==urn:ngsi-
ld:Store:001"}
    }
  },
  "notification": {
    "http": {
      "url": "http://tutorial:3000/subscription/low-stock-store001"
    },
    "attrsFormat" : "keyValues"
  }
}'
```

Figura 13: esempio con aggiunta del campo expression

In Figura [Fig.13], il meccanismo è simile al precedente ma differisce per la presenza dell'attributo "***expression***". L'esempio in Figura [Fig.13] fa riferimento alla gestione di un magazzino, in particolare quando la quantità di un prodotto su uno scaffale scende al di sotto di una certa soglia. La subscription viene controllata ogni volta che "*shelfCount*" di "*InventoryItem*" viene aggiornato e, eventualmente, questa viene eseguita se la condizione dell'attributo "*expression*" è verificata. Tale controllo viene effettuato nel campo "expression", all'interno di "condition" attraverso l'attributo "q". Nell'esempio in Figura [Fig.13], tramite l'attributo "q" si verifica se "shelfCount" è minore di 10 unità nel negozio (store001). Ciò significa che si può implementare la logica desiderata senza che le altre entità (quindi tutte le entità diverse da store001) ricevano la notification.

1.4 Mappa dei FIWARE Generic Enablers

Come descritto nei paragrafi precedenti, il componente principale in ogni applicazione "Powered by FIWARE" è il FIWARE Context Broker Generic Enabler. A tal proposito, il Fiware Catalogue offre un insieme ricco e sviluppato di blocchi software che possono essere assemblati, insieme al Context Broker, per velocizzare lo sviluppo di Smart Solutions. In particolare, queste componenti aggiuntive permettono di:

- Processare, analizzare e visualizzare le context information.
- Gestire, pubblicare e monetizzare i context data, tramite l'implementazione del comportamento intelligente desiderato e/o tramite assistenza verso utenti finali nelle loro Smart Solutions.

- Definire interfacce per la connessione con l'Internet of Things, sistemi robotici e di terze parti.
- Utilizzare Deployment Tools, dato che la maggior parte delle componenti FIWARE sono disponibili come immagini Docker.

Di seguito, si riporta uno schema dettagliato di tutti i Generic Enablers aggiuntivi al Context Broker [Fig.14]:

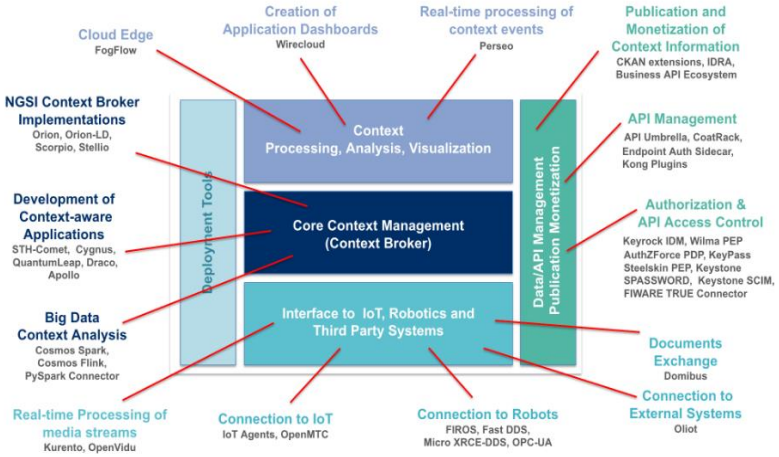

Figura 14: Schema dettagliato delle componenti aggiuntive al Context Broker

Uno dei vantaggi nell'utilizzare una soluzione "Powered by FIWARE" è che non si è costretti ad impiegare tutte le componenti presenti nel FIWARE Catalogue. Infatti, a partire dal Context Broker, è possibile aggiungere quelle di cui si necessita.

1.5 Strumenti per lo sviluppo di una Smart App

Una generica applicazione "Powered by FIWARE" può essere rappresentata attraverso un'architettura come quella proposta in Figura. [Fig.15]

Figura 15: Schema di funzionamento del Context Broker Orion

Questa applicazione fa utilizzo di un context Broker (*nell'esempio è l'Orion Context Broker*), di un database (*MongoDB*) per la persistenza di informazioni come context entities e di un software (*Postman*) per l'invio/ricezione. Un'evoluzione dell'architettura

precedente, che dimostra, inoltre l'interoperabilità dei blocchi FIWARE, è rappresentata dall'esempio che segue: [Fig.16]

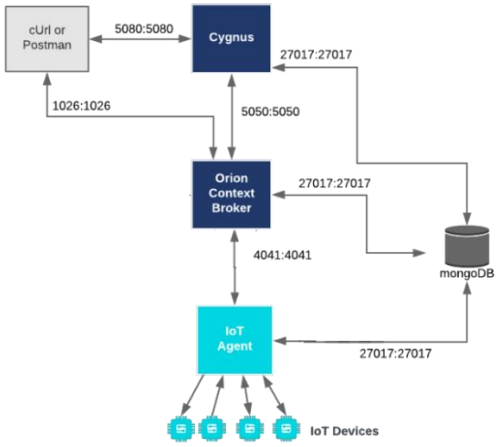

Figura 16: Esempio di architettura "Powered by FIWARE" ampliata

Nello schema in Figura [Fig.16], si utilizzano agenti IoT per ricevere dati da una serie di sensori, tramite un protocollo comune. L'agente IoT deve interpretare i dati e creare/modificare le entità NGSI nel Context Broker. Il sistema deve reagire ai cambiamenti di stato, pertanto viene utilizzato un GE (*Cygnus*) che garantisce persistenza nel database. Per velocizzare e facilitare lo sviluppo di una Smart App, esistono molteplici soluzioni software tra cui: Docker, MongoDB, Grafana, Postman e Node-RED, di seguito descritti.

1.5.1 Docker

Docker è una piattaforma software che permette di creare, testare e distribuire applicazioni con estrema velocità. Infatti, raccoglie il software in unità virtuali standardizzate chiamate "***container***", che offrono tutto il necessario per la loro corretta esecuzione, incluse librerie, tools di sistema, codice ed esecuzione runtime. In questo modo, il software è quindi ***system agnostic*** poiché non conta dove si trovi o su quale macchina stia girando. Docker permette di distribuire il codice più rapidamente e cerca di standardizzare il funzionamento delle applicazioni, inoltre garantisce l'esecuzione affidabile delle applicazioni in un container da qualsiasi posizione si acceda.

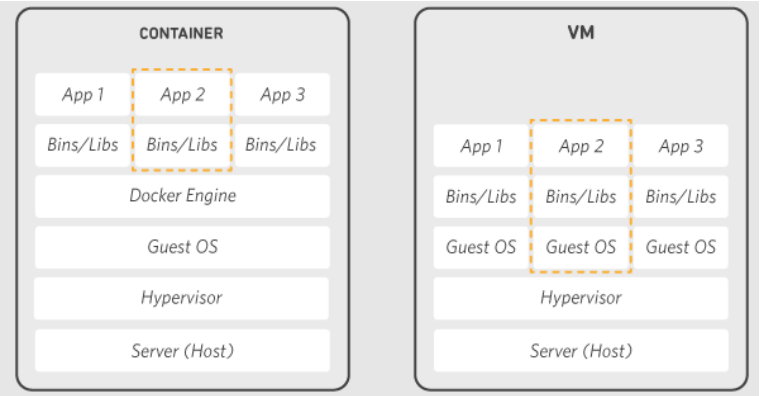

Figura 17: Differenza Container e Virtual Machine

Uno dei vantaggi di "localizzare" un'applicazione in un container sta nel fatto che l'identificazione di problemi o eventuali roll-back per il ripristino, vengono effettuate minimizzando costi e risorse. Così come una VM (Virtual Machine) virtualizza i server hardware, i container virtualizzano il sistema operativo di un server. Docker fornisce comandi semplici per creare, avviare o interrompere i container. Dal punto di vista tecnico, per costruire un container, si comincia da uno speciale file, in formato testuale, chiamato "*Dockerfile*". Questo consente di generare una Docker Image che verrà poi eseguita come un Docker container.

1.5.2 MongoDB

Anche nello sviluppo della più semplice applicazione intelligente, c'è bisogno di un database dal quale accedere/prelevare informazioni. Tuttavia, come si è analizzato precedentemente, i dati in gioco sono strutturati e complessi, per tale motivo, è necessario l'utilizzo di un database non relazionale: MongoDB. Quest'ultimo è un database NoSQL open source, altamente flessibile e che consente di combinare e memorizzare più tipi di dati. Infatti, memorizza i dati in documenti JSON-like, il che significa che i campi possono variare da documento a documento, ed è possibile modificare nel tempo la struttura dei dati. I vantaggi principali nell'utilizzo di un database di questo tipo sono:

- **Flessibilità:** MongoDB ha un'architettura a schema dinamico. Poiché i dati sono memorizzati in documenti JSON-like, lo schema del database non deve essere predefinito e può essere modificato nel tempo.

- **Partizionamento:** Tramite il partizionamento viene offerta scalabilità orizzontale. In questo modo, divide i dati di grandi dimensioni e li distribuisce su più server (o container).

- **Performance Elevate:** MongoDB memorizza i dati nella RAM per accelerarne l'accesso e ottimizzare l'esecuzione delle query, rendendo più veloce letture e scritture. La sua struttura non relazionale richiede una minore potenza di elaborazione per la ricerca e il recupero dei dati, rispetto a quanto avviene in un database relazionale.

1.5.3 Grafana

Grafana è una piattaforma open source per la visualizzazione grafica dei dati, sviluppata da Grafana Labs, che permette agli utenti di visualizzare i dati attraverso diagrammi e grafici unificati in una o più dashboard, al fine di agevolarne la comprensione e l'interpretazione. Come gli altri tools, Grafana semplifica lo studio di dati provenienti da sorgenti diverse.

1.5.4 Postman

Postman è un ambiente di sviluppo API completo che consente agli sviluppatori di effettuare richieste API e ispezionare i dati di invio e ricezione. Supporta vari tipi di richieste come REST, SOAP, GraphQL. Inoltre, Postman gestisce formati di risposta come JSON, XML, HTML e testo semplice. Una funzionalità molto utile di Postman è quella del "Server simulato" che consente di simulare le risposte dell'API pur non avendo un server fisico.

1.5.5 Node-RED

Node-RED è un linguaggio di programmazione a blocchi, sviluppato per gestire in modo semplice i flussi di dati che viaggiano nell'ambienti IoT, o generati tramite sensori. Esso fornisce un editor di flussi basato su browser e una collezione di nodi per la creazione degli stessi dinamicamente. Si basa sul protocollo MTTQ (Message Queuing Telemetry

Transport) di IBM, ovvero lo standard di riferimento della comunicazione nel mondo IoT. Le informazioni tra i nodi di ogni flusso viaggiano come messaggi (*msg*) alfa-numerici o di tipi JSON e sono contenuti nei blocchi *payload.* Quindi, ogni messaggio trasmesso ad un nodo del flusso può essere inviato a quello successivo e così via, al fine di realizzare l'operazione desiderata.

1.6 Principali domini applicativi di FIWARE

In questa sezione, vengono proposte le aree più rilevanti in cui FIWARE viene adoperato. In particolare:

- **Digital Twin**
- **Data Space**
- **Smart City**

Per gli ultimi due, si propone di seguito una descrizione del contesto applicativo e dell'architettura di riferimento, invece, i Digital Twin verranno affrontati nel Capitolo 2 tramite l'illustrazione di casi d'uso.

1.6.1 FIWARE for Data Spaces

Nel mondo informatico, il concetto di *Data Space* (DS) è stato coniato circa 15 anni fa come concetto di integrazione dei dati [1]. A differenza degli approcci di integrazione dati centralizzati, come i data consolidation hub per la rimozione di records duplicati in un database, i DS non richiedono un'integrazione fisica dei dati ma li lasciano archiviati alla fonte, consentendo così ridondanze e "coesistenza" di dati. Oltre alla definizione tecnica appena illustrata, il continuo utilizzo del termine nella comunità imprenditoriale ha portato a concepire il concetto di DS come una forma di collaborazione sui dati. *Open DEI* cioè, "*Open Platforms and Large-Scale Pilots in Digitizing European Industry"* un progetto europeo per il supporto della trasformazione digitale in Europa, ha definito formalmente uno *spazio dati* come: "un'infrastruttura decentralizzata per la condivisione e lo scambio affidabile di dati in ecosistemi basati su questi, tramite regole comunamente concordate". Mentre Gaia-X, progetto europeo con lo scopo di promuovere lo scambio dati tra aziende,

enti di ricerca ed enti pubblici nel rispetto della proprietà di questi, ha definito un DS come: “uno spazio dove partecipanti fidati, che aderiscono agli stessi standard e linee guida, possono condividere dati in piena sicurezza”. Da queste definizioni, si comprende che i Data Spaces e le infrastrutture software che li supportano, devono garantire sicurezza, interoperabilità e portabilità dei dati. In quest’ottica, risulta fondamentale la condivisione di dati in un Data Space. Per questo motivo, oltre a Gaia-X, esiste un’altra nota associazione ovvero: *IDSA (International Data Spaces Association*), fondata nel 2017 e senza scopo di lucro, per creare standard di condivisione dati tra i suoi partecipanti e garantire meccanismi di controllo e tracciabilità. Si tratta di una delle principali iniziative che definiscono le specifiche tecniche per gli spazi dati nell’Unione Europea. Oltre a certificare componenti software utilizzati nei Data Spaces, ne fornisce un’architettura di riferimento [Fig.18].

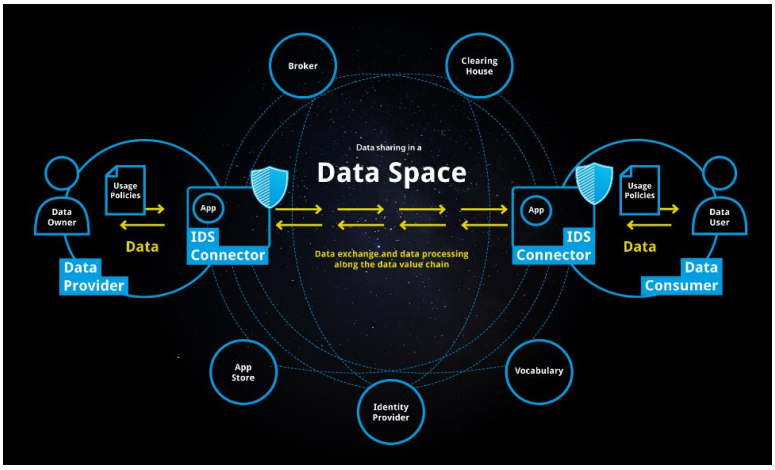

Figura 18: Architettura di riferimento per un Data Space

Quindi, gli spazi dati devono supportare meccanismi per la pubblicazione e scambio di dati, elementi che sono implementati da FIWARE. Infatti, queste componenti possono essere combinate con altre, come il “*IDS Connector*” dell’architettura di IDSA, per la creazione di spazi dati sicuri e che garantiscono uno scambio dati efficace. Di seguito, viene illustrato un esempio di integrazione tra FIWARE e il IDS Connector. [Fig.19]

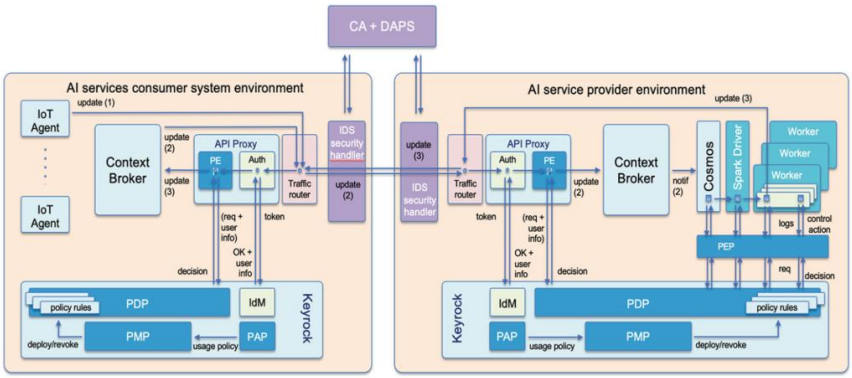

Figura 19: Esempio di integrazione tra FIWARE e IDS Connector

La Figura [Fig.19] illustra come due servizi AI, un provider e un consumer, interagiscono tra loro. In particolare, ognuno dei servizi appartiene a organizzazioni diverse ma comunicano, come in questo caso, partecipando ad un Data Space comune "Powered by FIWARE". Il servizio provider potrebbe, per esempio, essere un servizio avanzato per l'analisi predittiva del traffico mentre, il consumer, potrebbe il sistema di gestione del traffico di una data città.

1.6.2 FIWARE for Smart Cities

Con il crescente aumento della popolazione urbana, particolare attenzione è stata rivolta alla creazione di ambienti in cui le persone e le imprese possano svilupparsi in modo *smart*. Infatti, molte città si stanno avvicinando al concetto di "Smart City" per rendere più efficienti infrastrutture e servizi già esistenti. Con FIWARE, le città possono realizzare questa trasformazione attraverso l'adozione di API standard e Data Model comuni. I vantaggi di una "Smart City" sono:

- Miglioramento dei sistemi di governance, per esempio la sanità, la sicurezza e l'istruzione.
- Azzeramento costi di adattamento: grazie a modelli informativi standard "de facto", si raggiunge la completa interoperabilità e portabilità tra i diversi sistemi di una città.
- Open data pubblicati in tempo reale da una città: tali dati sono resi disponibili tramite API standard e i fornitori di Smart Solutions possono adoperarli, al fine di vendere tali soluzioni alle città di tutto il mondo.

Quindi, in una Smart City giocano un ruolo cruciale i sensori IoT per la raccolta dati, i software di elaborazione di dati storici e real-time per estrarre informazioni e dashboard per il monitoraggio dei cambiamenti in una città e verifica degli obiettivi prefissati. Una Smart City risulta, quindi, essere un esempio di "*System of systems"* (SoS) ovvero un insieme di "sistemi" indipendenti ma interagenti che, insieme, formano un sistema complesso e unificato. A tal proposito, FIWARE offre un'architettura di riferimento per lo sviluppo di soluzioni verticali (ovvero soluzioni che rispondono a specifiche esigenze) per una Smart City, in inglese nota come "*FIWARE Reference Architecture for Smart City vertical solution*". [Fig.20]

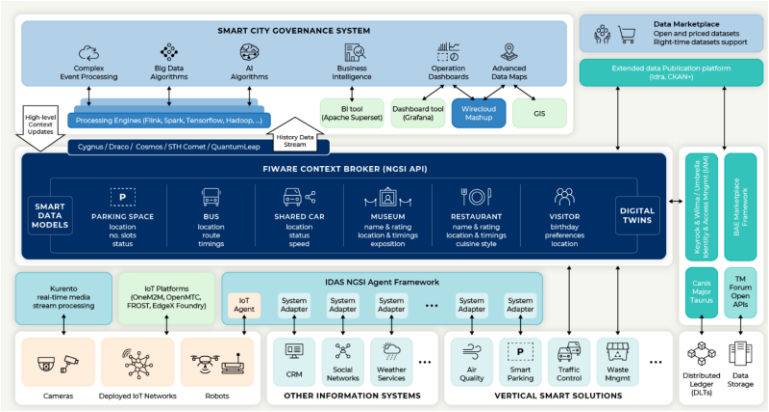

Figura 20: FIWARE Reference Architecture for Smart City vertical solution

Come si può analizzare dalla Figura [Fig.20], differenti risorse di dati (mobilità, sistemi per il controllo della qualità dell'aria e altri) possono essere integrate con sistemi già esistenti al fine di fornire informazioni utili in tempo reale in modo comprensibile e unificato. Riferendosi all'esempio di ecosistema FIWARE in Figura [Fig.20], si ritrovano elementi già trattati precedentemente, quali:

- il Context Broker, contenente la rappresentazione Digital Twin dell'intera città. Tutte le interazioni tra applicazioni o componenti e il Context Broker avvengono tramite l'API NGSI-LD.
- Al di sotto del Context Broker, sistemi di acquisizione dati quali: agenti IoT NGSI che possono essere connessi con dispositivi IoT (sensori/telecamere) e piattaforme IoT di terze parti per lo sviluppo di interfacce verso sistemi robotici o di sorveglianza.

- Al di sopra del Context Broker, sistemi per la gestione, analisi e visualizzazione dei dati storici e real-time, come Grafana, e una serie di componenti FIWARE quale Cygnus.
- A destra, sono mostrate alcune componenti FIWARE relative alla sicurezza, come Keyrock o AuthZForce, per garantire che i dati siano accessibili solo a utenti e applicazioni autorizzate.

FIWARE cerca da anni di portare sul campo la propria visione di Smart City, tanto è che, ad oggi, conta più di 300 città in più di 30 paesi nel mondo, che sono parte dell'ecosistema "FIWARE for Smart Cities". In particolare, nel territorio italiano, Firenze, Milano, Perugia, Genova e Roma sono esempio di Smart Cities "Powered by FIWARE".

Capitolo 2: FIWARE for Digital Twin

In questo capitolo verranno analizzati due casi di utilizzo di FIWARE per i Digital Twin. Preliminarmente, visto che sono state formulate molteplici definizioni per i DT, si cercherà di fornirne una generale ed eventuali differenze da altre tecnologie digitali simili. In una fase successiva, verranno discusse le due architetture in cui si utilizza FIWARE a supporto dei Digital Twin.

2.1 Il concetto di Digital Twin

Il termine "Digital Twin" venne utilizzato, per la prima volta, in una presentazione nel 2003 sulla gestione del ciclo di vita di un prodotto, da parte del Dr. Micheal Grieves (ad oggi direttore esecutivo del *Digital Twin Institute*) e John Vickers [3] (capo tecnico della NASA nel campo della modellazione di sistemi aerospaziali all'avanguardia). L'idea di Grieves e Vickers era quella di poter passare da un prodotto prevalentemente manuale ad un modello digitale dello stesso al fine di migliorarne l'efficienza dal punto di vista del ciclo di sviluppo. Mentre concetti simili quali *CPS* (*Cyber Physical Systems*) e l'*IoT* (*Internet of Things*) si basano sull'idea di connettere un sistema fisico a sistemi di raccolta dati (CPS da punto di vista dell'ingegneria di sistema e IoT dal punto di vista del networking e dell'IT), i Digital Twin si pongono l'obiettivo di connettere un sistema fisico ad uno digitale dal punto di vista della modellazione computazionale, tramite apprendimento automatico e utilizzo dell'AI. Secondo la prima caratterizzazione del concetto di Digital Twin di Grieves, questo consisteva in tre componenti: un prodotto fisico nello spazio reale, una rappresentazione di quel prodotto nello spazio virtuale e un

insieme di connessioni di dati e informazioni che legano il prodotto reale e quello virtuale, quest'ultimo interpretabile come un sistema di aggiornamento attraverso scambio di informazioni. In base a tale caratterizzazione [3], i processi e le componenti di un Digital Twin possono essere schematizzati come illustrato nella seguente Figura [Fig.21].

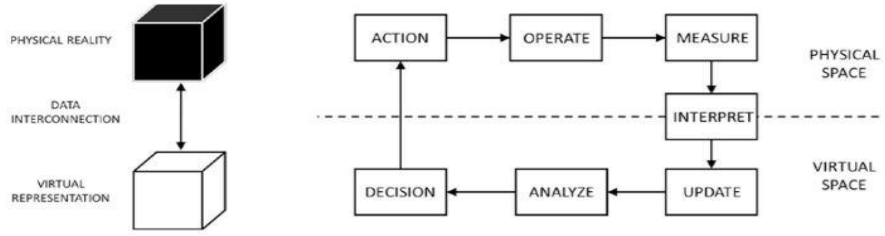

Figura 21: Schematizzazione componenti Digital Twin e relativi processi

Nella prima fase, oltre alla raccolta dati manuale e offline, si fa riferimento al mondo dell'IoT e dei sensori. Sebbene non siano fondamentali per lo sviluppo di un DT, queste tecnologie risultano essere estremamente utili poiché consentono di raccogliere un elevato quantitativo di informazioni.

Figura 22: Processo di interpretazione

La seconda fase del processo riguarda *l'interpretazione* dei dati raccolti [Fig.22]. Questa fase risulta essere il punto di separazione tra lo spazio fisico e lo spazio virtuale. Infatti, a causa dell'astrazione della realtà fisica, i dati raccolti devono essere trasformati in dati "comprensibili" per la corrispettiva rappresentazione virtuale tramite elaborazione e conversione. L'ultima fase riguarda, invece, l'utilizzo di tali dati per aggiornare gli stati della rappresentazione virtuale. Nei casi più semplici, i dati misurati tramite sensori corrispondono, in modo esatto, ad uno stato nella rappresentazione virtuale. Infatti, nei Digital Twin, il prodotto virtuale deve necessariamente mutare il proprio stato così come farebbe la sua controparte reale. Nei casi più complessi, invece, l'aggiornamento di stato potrebbe essere ottenuto tramite tecniche più avanzate, per esempio quando i dati raccolti provengono da più fonti. Sebbene questa sia la definizione più generale di un Digital

Twin, in letteratura è presente un'ulteriore classificazione [3] degli stessi in base al flusso di dati tra prodotto fisico e virtuale. In Figura [Fig.23] viene illustrata tale classificazione.

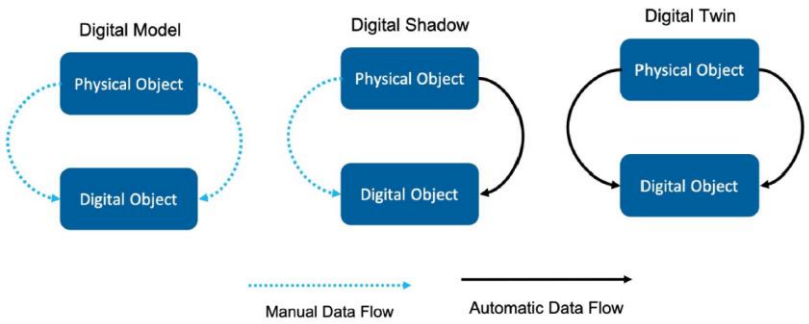

Figura 23: Digital Model, Digital Shadow e Digital Twin con relativi flussi dati

In particolare, ci si riferisce con il termine di *Digital Model* quando il flusso di dati dall'oggetto fisico a quello virtuale è manuale, e viceversa. Viene chiamato, invece, *Digital Shadow* quando il flusso di dati è automatico dal mondo fisico a quello digitale, ma è manuale nell'altra direzione. Questo significa che eventuali cambiamenti di stato dell'oggetto fisico producono, in *automatico*, un cambio di stato dell'oggetto digitale, ma ciò non vale nella direzione opposta. Si parla, invece, di *Digital Twin* quando il flusso dati è automatico in entrambe le direzioni, quindi i cambiamenti di uno si riflettono sull'altro [3].

2.2 Sviluppo di Digital Twin con FIWARE

Come descritto in precedenza, il seguente lavoro di tesi offre una revisione di architetture di riferimento per la costruzione di Digital Twin, basata sui Generic Enablers di FIWARE e Smart Data Models. La Fondazione FIWARE definisce un DT come: "*un'entità che rappresenta digitalmente una risorsa fisica del mondo reale*". Il processo di *gemellaggio* dal mondo fisico a quello virtuale, noto in letteratura come *twinning* [3], inizia con un sensore IoT che rileva un cambiamento in un'entità fisica (*metrology phase*) e termina con l'aggiornamento delle informazioni di contesto (*realization phase*). Il processo inverso, ovvero il gemellaggio dallo spazio digitale a quello fisico, inizia con un aggiornamento di un'entità virtuale (*metrology phase)* e termina con un attuatore IoT che aggiorna lo stato

della corrispettiva entità fisica (*realization phase*). Per entrambi i processi, il FIWARE Catalogue offre Generic Enablers per agenti IoT (intesi come componenti software collocate tra i dispositivi IoT e il Context Broker) ovvero dei moduli per tradurre le richieste NGSI nei protocolli di comunicazione dei dispositivi IoT (per esempio, *UltraLight*) e viceversa. Risulta, quindi, evidente che una delle sfide dei DT è quella che riguarda l'acquisizione di dati da fonti esterne e i loro adattamento ai Smart Data Models. In FIWARE, una soluzione a tale problematica è offerta dal Draco GE, un sistema di gestione del flusso dati altamente scalabile e adatto per operazioni di trasformazione dati in un formato standard per favorire così l'analisi e l'integrazione. L'iniziativa FIWARE Smart Data Models fornisce, quindi, gli strumenti necessari per la definizione di uno schema comune e di modelli di dati standard, utili per facilitare l'integrazione dei DT con altri sistemi esterni. Tali strumenti saranno, in particolar modo, presentati e discussi attraverso l'analisi di due casi d'uso presenti in letteratura, *Parking DT* [4] e *Airport DT* [5], di seguito descritti.

2.3 Parking Digital Twin

Il presente caso d'uso permette di comprendere come progettare un DT nell'ecosistema FIWARE nel mondo del Parking [4]. L'obiettivo di questo DT è quello di migliorare la gestione di un parcheggio, aiutando i clienti a trovare il proprio veicolo tramite un'applicazione mobile che fornisce una vista 3D del parcheggio e previsioni sulla disponibilità di posti nello stesso in un dato giorno. Il sistema, quindi, raccoglie dati da sensori installati nel parcheggio e, inoltre, attraverso un API esterna, acquisisce le informazioni metereologiche per effettuare previsioni. Gli step per la progettazione di tale DT sono: modellazione, acquisizione, consumo dei dati e sicurezza. Di seguito in Figura [Fig.24], viene presentata un'architettura semplificata del sistema in questione.

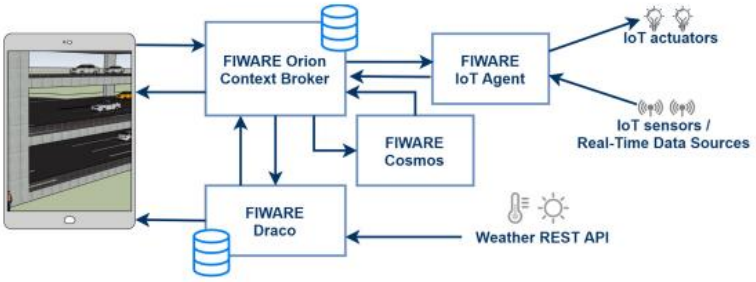

Figura 24: Architettura di riferimento per un Parking DT

2.3.1 Modellazione dei dati

Per quanto riguarda la fase di modellazione dei dati, in questo caso d'uso sono stati selezionati quattro differenti modelli dal catalogo degli *Smart Data Models*, per rappresentare i tipi di entità coinvolte, ovvero: *OffStreetParking, ParkingSpot, Vehicle e WeatherForecast*. Il primo tipo serve a modellare l'intero parcheggio, il secondo un singolo spazio per parcheggiare, il terzo è utilizzato per modellare un veicolo di un cliente e, infine, l'ultimo tipo per effettuare previsioni metereologiche per la prossima ora.

2.3.2 Acquisizione dei dati

I dati provengono da diverse sorgenti, pertanto è necessario adattare tali informazioni ai corrispettivi data models. I dati sono raccolti da sensori IoT che forniscono informazioni sui veicoli e i posti che questi occupano nel parcheggio. In questo esempio, i sensori utilizzano il protocollo di comunicazione *Ultralight 2.0.* Per esempio, quando un veicolo parcheggia in un determinato posto, viene generato un messaggio nel formato: < id | 123456 | t | car | p | 51 >. Tale messaggio rappresenta un'entità veicolo di tipo "car" con numero di targa "123456" che è parcheggiata nello slot numero "51". Il relativo agente IoT riceve questo messaggio, lo trasforma nelle entità secondo il formato NGSI v2 e aggiorna le context information mandando tre richieste HTTP al Context Broker Orion. Di seguito sono elencate le tre richieste in questione:

- La prima richiesta crea un'entità di tipo "Vehicle" che rappresenta l'automobile in questione [Fig.25].

```
{
 ''id'': ''vehicle:501,''
 ''type'': ''Vehicle,''
 ''vehicleType'': ''car,''
 ''vehiclePlateIdentifier'': ''123456''
}
```


Figura 25: Entità *Vehicle*

- La seconda richiesta aggiorna l'entità di tipo ParkingSpot, con stato (status) occupato e numero di posto pari a 51 [Fig.26]


```
{
 ''id'': ''spot:51,''
 ''type'': ''ParkingSpot,''
 ''name'': ''51,''
 ''status'': ''occupied,''
 ''refVehicle'': ''vehicle:501,''
 ''refOffStreetParking'': ''parking:1''
}
```


Figura 26: Aggiornamento entità ParkingSpot

- Infine, l'ultima richiesta decrementa l'attributo "availableSpotNumber" dell'entità OffStreetParking [Fig.27]


```
{
 ''id'': ''parking:1,''
 ''type'': ''OffStreetParking,''
 ''availableSpotNumber'': 1449
}
```


Figura 27: Decremento contatore per i posti disponibili

L'ultima fonte dati esterna riguarda l'entità di tipo *WeatherForecast.* Il Generic Enabler di FIWARE Draco preleva periodicamente i dati dall'API esterna per le previsioni metereologiche e li adatta al data model dell'entità *WeatherForecast,* trasformando il dato in JSON in formato NGSI v2. Infine, tramite una richiesta HTTP ad Orion, aggiorna le informazioni di contesto.

2.3.3 Utilizzo dei dati

Dato che il Context Broker contiene le versioni più recenti delle context information per le entità che gestisce, le altre componenti ad esso connesse effettuano una *subscription* al fine di essere aggiornate su eventuali cambiamenti. Pertanto, l'applicazione mobile in questione effettuerà una richiesta HTTP POST ad Orion per ricevere aggiornamenti circa l'attributo *status* delle entità *ParkingSpot*. Quindi, nel momento in cui un posto viene occupato o liberato, una notifica sarà mandata all'applicativo per segnalare tale cambiamento. Inoltre, un'ulteriore notifica viene mandata anche ad agenti IoT. In particolare, l'agente IoT riceve una *notification,* la elabora e invia un'azione da eseguire ovvero l'aggiornamento dello stato di un led installato in tutti gli slot per parcheggiare. In particolare, quando un posto viene occupato, lo stato del led cambia da verde (libero) a giallo (occupato) e viceversa. Anche l'altro GE coinvolto, COSMOS [Fig.24], effettua una *subscription* al Context Broker, in particolare per l'entità *WeatherForecast* e l'attributo *status* delle entità *ParkingSpot*. Utilizzando questi dati e altre informazioni, per esempio ora e giorno della settimana, possono essere implementati algoritmi di machine learning al fine di effettuare previsioni in tempo reale sullo stato del parcheggio e medie, ad esempio, per stimare il numero di veicoli che utilizzano il servizio in un certo giorno. A tale scopo, Draco salva tutti i cambiamenti circa le context information in un database Mongo, per l'accesso a dati recenti e passati.

2.3.4 Generic Enablers per la sicurezza

Un aspetto fondamentale dei DT è sicuramente quello che riguarda la confidenzialità e l'integrità dei dati. Risulta, quindi, necessario gestire ruoli e permessi per garantire che si acceda in modo sicuro alle informazioni contenute nel Context Broker. A tal fine, FIWARE fornisce diversi GEs in questo campo e, in questo esempio, quelli utilizzati sono: FIWARE KeyRock e Wilma. In Figura [Fig.28], viene presentata l'architettura iniziale, arricchita dei due GEs di sicurezza appena citati.

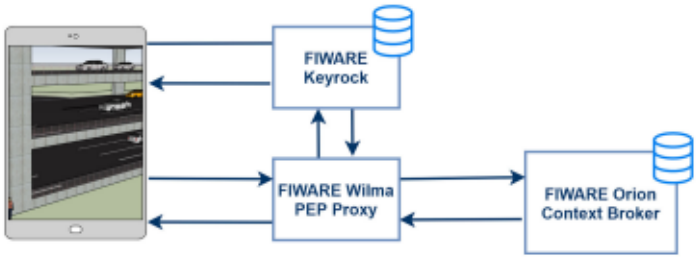

Figura 28: Architettura con aggiunta dei GE per la sicurezza

Focalizzando l'attenzione sull'accesso all'applicazione mobile, si possono distinguere tre tipi di ruoli: *admin*, *supervisore* e *utente*. Inoltre, ci sono tre permessi di base quali: creazione e cancellazione dei supervisori, aggiornamento dello stato delle entità *ParkingSpot* e recupero di queste. Per esempio, all'*admin* sono forniti tutti e tre i permessi, al supervisore solamente il permesso di aggiornamento e recupero delle entità *ParkingSpot*. All'utente è permesso solamente poter visualizzare lo stato corrente del parcheggio. Questi ruoli e permessi sono creati e assegnati dall'amministratore tramite l'interfaccia di KeyRock. Dall'altra parte, Wilma permette di gestire in sicurezza gli accessi al Context Broker, nascondendo gli indirizzi reali ad utenti esterni. Infatti, gli utenti non interagiscono direttamente con Orion, ma mandano le proprie richieste al GE Wilma. Dopo aver verificato se gli utenti hanno o meno i permessi necessari, Wilma ridireziona tali richieste ad Orion.

2.4 Airport Digital Twin

Il caso d'uso di seguito è applicato al settore aerospaziale, dove i Digital Twin sono stati impiegati sin dall'origine. Nell'industria aerospaziale, i Digital Twin sono stati utilizzati per produrre parti di aerei o per verificare la presenza di eventuali difetti di produzione. In letteratura esistono studi che cercano di estendere l'utilizzo dei DT ad altri scopi come: manutenzione, rilevamento guasti, monitoraggio prestazioni di un elicottero o per la previsione di danni durante un volo. Il presente caso d'uso, in particolar modo, si basa sull'implementazione di un Digital Twin presso l'aeroporto internazionale di Aberdeen [5] (Scozia) per monitorare e gestire gli eventi di *turnaround* di un velivolo, inteso come l'insieme di attività svolte in aeroporto per preparare un aereo in arrivo per il successivo

volo. Tramite la gestione efficiente di questo processo, il Digital Twin viene impiegato per ridurre ritardi nei voli. Nei paragrafi successivi, seguirà l'analisi del caso d'uso attraverso la descrizione dei seguenti passaggi quali: presentazione dell'architettura di riferimento, definizione di Data Models nel dominio d'interesse e, infine, gestione e visualizzazione dei dati.

2.4.1 Architettura di riferimento

In Figura [Fig.29], viene mostrata l'architettura di riferimento per l'Airport DT [5].

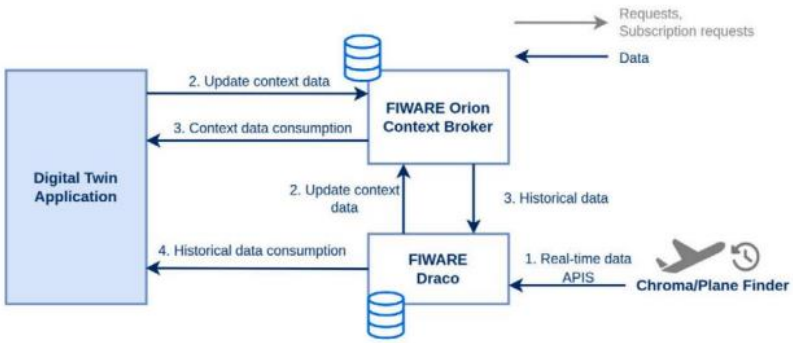

Figura 29: Architettura per l'Airport Digital Twin

Come in ogni applicazione "*Powered by FIWARE*", l'elemento centrale è il Context Broker (in Figura è riportato come *FIWARE Orion Context Broker*) che ha il compito di fornire funzionalità per la gestione del contesto, utilizzando NGSI come standard. Una prima differenza con il caso d'uso del paragrafo precedente è l'utilizzo dello standard NGSI-LD (si ricorda che LD sta per Linked Data) rispetto a quello NGSI v2. Pertanto, tutti i dati rappresentati come entità nel Context Broker devono essere conformi a NGSI-LD e, a tal fine, il Generic Enabler Draco consente di effettuare questa conversione dei dati in ingresso in entità e attributi. Infine, sono utilizzate due applicazioni client, dove: la prima è un'applicazione web che permette di visualizzare una rappresentazione real-time dell'aeroporto attraverso modelli 2D e 3D, mentre la seconda è un'applicazione mobile che agevola il compito degli operatori durante le operazioni di turnaround.

2.4.2 Definizione di Data Models

Il presente caso d'uso appartiene al dominio applicativo noto come *SmartAeronautics.* Il Data Model di riferimento è il *Flight data model.* Per semplicità di notazione, vengono illustrate brevemente le sole entità utili alla comprensione del caso d'uso, tralasciando notazioni strettamente specifiche del domino aerospaziale. Utilizzando il modello dati in questione, un'entità di tipo *volo,* identificata da un *numero di volo,* è rappresentata come l'insieme di attività legate ad un volo dall'istante di partenza fino a quello di arrivo. Un'entità di tipo *aereo* rappresenta il velivolo fisico che opera durante un *volo,* identificato da un *numero di coda*. Inoltre, viene modellata un'entità di tipo *FlightNotification,* per la registrazione di un qualsiasi evento durante un volo. Infine, viene modellata l'entità di tipo *Airport* per la rappresentazione di un aeroporto reale, con un proprio identificativo e attributi, per esempio: posizione, indirizzo e nome. Tutti i data models qui presentati seguono le linee guida definite nel ciclo di vita di un qualsiasi Smart Data Model [Cap. 1 par. 1.3].

2.4.3 Gestione dei dati

Una volta definiti i data models utilizzati nel dominio di interesse, le applicazioni e i servizi sono in grado di ottenere dati standardizzati con una struttura omogenea. Tuttavia, come già sottolineato, una delle principali sfide nell'utilizzo dei Digital Twin riguarda il processo di ottenimento di dati da differenti sorgenti e il loro adattamento con i modelli definiti. Nel caso d'uso del Parking DT [Par.2.3], i dati sono ottenuti da sensori IoT e adattati poi dal Draco GE. Anche nell'architettura del presente caso di studio viene impiegato Draco per lo stesso scopo, ma è completamente dipendente dalle sorgenti di dati utilizzate. In particolare, sono due le sorgenti dati impiegate per ottenere i dati in questione: l'API *Chroma* e *Plane Finder*. Chroma fornisce al Digital Twin informazioni su voli, aeroporti e compagnie aeree. Il servizio fornisce un'interfaccia che permette al GE Draco di stabilire periodicamente una connessione. Una volta stabilita, le informazioni vengono inviate a Draco attraverso una serie di processori configurabili e riutilizzabili per eseguire semplici operazioni di adattamento dei dati nel formato specificato dal data

model. Tra questi processori è utile citare il *JoltTransformJson_To_NGSI.* Tale richiesta trasforma l'entità di volo in ingresso per renderla compatibile con lo standard NGSI-LD. Quindi, un dato ottenuto tramite Chroma è trasformato in un'entità NGSI-LD di tipo *volo.* In Figura [Fig.30], sono illustrati i processori coinvolti nella cattura dati dall'API Chroma.

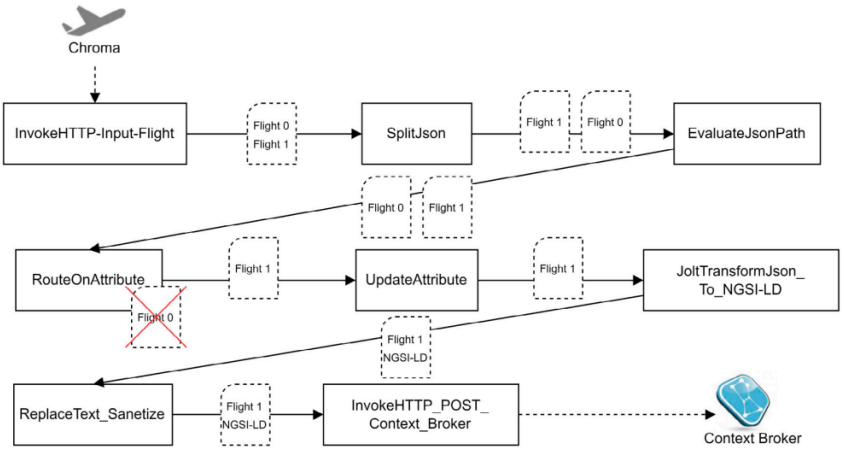

Figura 30: Cattura dati da Chroma

Si noti che vengono effettuate anche operazioni di controllo (*ReplaceText_Sanetize*) per evitare caratteri non supportati dal formato NGSI-LD e di salvataggio (*InvokeHTTP_POST_ContexBroker*) per l'inserimento dell'entità nel Context Broker. L'altra sorgente dati è Plane Finder che fornisce al DT informazioni riguardo la posizione degli aerei nelle prossimità dell'aeroporto. Anche in questo caso, Draco trasforma i dati ricevuti da Plane Finder in un formato compatibile allo standard NGSI-LD. Vengono coinvolti i medesimi meccanismi, esattamente come illustrato nella precedente Figura [Fig.30]. Pertanto, il Generic Enabler di FIWARE mostra un comportamento altamente robusto nell'affrontare problemi di ottenimento dati da sorgenti diverse e adattamento degli stessi, tramite una serie di processori che implementano semplici funzioni con il vantaggio che questi sono altamente configurabili. Infine, la soluzione qui presentata fornisce il vantaggio di accedere ai dati storici. Dato che il Context Broker salva unicamente lo stato corrente ovvero l'ultimo aggiornamento di stato delle entità che tratta, i dati storici sono gestiti da Draco, che li salva in un database Mongo.

2.4.4 Comunicazione tra Context Broker e Consumatori

Viene di seguito riportato un approfondimento su come avviene la comunicazione tra il Context Broker ed eventuali consumatori (servizi esterni, applicazioni). Distinguiamo due tipi di comunicazione: *sincrona* e *asincrona*. Per comunicazione sincrona si intende quel tipo di comunicazione in cui i client inviano *direttamente* richieste HTTP al Context Broker per creare, leggere, aggiornare o eliminare entità (in questo caso, per ottenere informazioni sui voli o ricevere eventuali notifiche). Nell'altro tipo di comunicazione, quella asincrona, i client, che hanno in precedenza effettato una subscription al Context Broker, ricevono notifiche HTTP quando una o più entità (o attributi di queste) a cui sono sottoscritti cambia. In questo caso, le operazioni più comuni possono essere: ottenere un volo dato il suo identificativo, filtrare un volo per data, ottenere la posizione di un aereo o attivare/disattivare notifiche per un volo. Anche il DT del parcheggio presentava lo stesso tipo di meccanismo ma, mentre nel primo caso d'uso era stata implementata un'applicazione mobile per la gestione del parcheggio, qui viene presentato un sistema di gestione delle sole operazioni di turnaround. In Figura [Fig.31], viene illustrato un esempio di comunicazione asincrona.

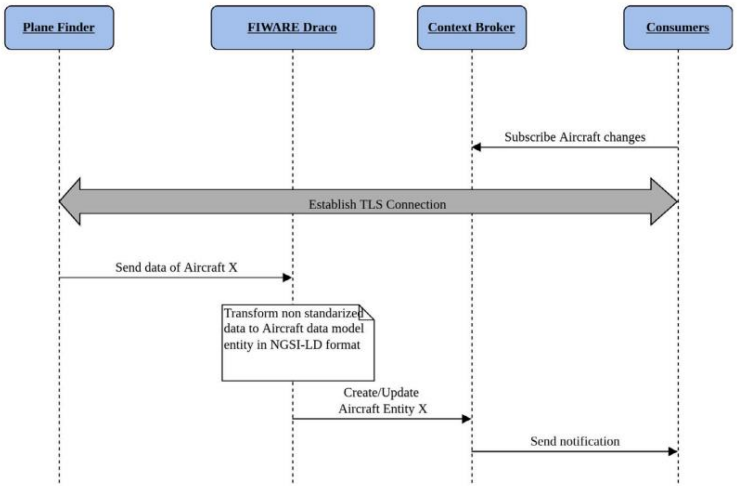

Figura 31: Esempio di comunicazione asincrona

2.4.5 Visualizzazione dei dati

In questa sezione viene illustrato brevemente come i dati gestiti dall'architettura FIWARE presentata all'inizio sono utilizzati dall'Airport DT. Al fine di ridurre eventuali ritardi nei voli, vengono di seguito citate due applicazioni web che, comunicando con il resto dell'architettura del DT, facilitano lo svolgimento delle operazioni di volo. La prima applicazione mostra lo stato dell'aeroporto in tempo reale, combinando sia la rappresentazione 2D che quella 3D (VR). La seconda, invece, è destinata agli operatori di terra per la registrazione e lettura di eventi durante la fase di turnaround.

Conclusioni

Il presente lavoro di tesi ha avuto lo scopo di illustrare cos'è FIWARE e l'intero ecosistema che lo compone, tramite la presentazione delle sue componenti fondamentali come il Context Broker, l'analisi degli standard e dei formati necessari per il progetto e sviluppo di Smart Solutions in diversi settori applicativi. A tale fine, sono stati presentati esempi nel campo *FIWARE for Digital Twin*, fornendo una definizione generale del concetto di gemello digitale e illustrando due architetture di riferimento tratte dalla letteratura. Sulla base dei casi di studio presentati, si è visto come FIWARE, i suoi Generic Enablers e i FIWARE Smart Data Models costituiscono un punto di riferimento altamente valido per lo sviluppo di Smart Solutions, implementate anche con tecnologie complesse come i Digital Twin. In più, essendo i GEs di FIWARE open source, la loro evoluzione e capacità di adattamento a nuove tecnologie e standard è assicurata dalla sempre crescente community di FIWARE. In entrambi i casi presentati, emergono i punti di forza della piattaforma FIWARE, la capacità di adattamento a differenti approcci ed eventuali miglioramenti che possono interessarla. Infatti, in riferimento al caso d'uso del Parking DT con FIWARE, si potrebbe proporre come sviluppo futuro, un adattamento dell'architettura allo standard NGSI-LD per i Linked Data oppure testare altri Context Broker alternativi ad Orion e verificare eventuali cambiamenti. Nel caso dell'Airport DT, invece, l'architettura e le metodologie presentate risultano essere più complesse rispetto al caso precedente, vista anche la natura strettamente tecnica del dominio aerospaziale. Nello specifico, l'architettura proposta trae vantaggio dal GE Draco, dal Context Broker Orion, dallo standard NGSI-LD e dall'utilizzo di Smart Data Models per fornire dati in tempo

reale e accesso a quello storici. Sicuramente, FIWARE promuove l'*interoperabilità* tra diversi sistemi e dispositivi, facilitando l'integrazione. Essendo una piattaforma open source, FIWARE consente agli sviluppatori di accedere liberamente alle componenti messe a disposizione e adattarle alle proprie esigenze specifiche, riducendo i costi e promuovendo l'innovazione attraverso la propria community. Un altro punto di forza di FIWARE è l'utilizzo di standard aperti, facilitando in questo modo la scalabilità e la replicabilità di soluzioni in diversi contesi applicativi. Come illustrato in questo lavoro di tesi, FIWARE offre una vasta gamma di componenti riutilizzabili, i Generic Enablers, che coprono diversi aspetti dello sviluppo di Smart Solutions, dalla gestione dati alla sicurezza. Infine, essendo progettata per supportare l'IoT, la piattaforma FIWARE facilita la connessione e la gestione di un ampio numero di dispositivi IoT, consentendo così la creazione di soluzioni intelligenti di alta qualità. Tuttavia, sebbene i vantaggi sopra elencati, c'è da tenere in conto alcuni punti a sfavore. La complessità, le risorse necessarie per la manutenzione e il fatto di essere in continua evoluzione sono fattori da tenere in conto nella scelta della piattaforma. Implementare e mantenere una soluzione basata su FIWARE può richiedere risorse tecniche e umane significative e le organizzazioni devono essere pronte a investire in personale qualificato e infrastrutture adeguate a garantire efficienza e qualità. Per concludere, un aspetto estremamente attuale che interessa la piattaforma è il rapporto tra FIWARE e temi come Big Data e Machine Learning, esenti dal presente lavoro di tesi. A tal proposito, FIWARE si sta integrando sempre più con queste tecnologie per migliorare la capacità di analisi e predizione dei dati. Infatti, nella maggior parte dei casi, architetture già esistenti basate su FIWARE riescono ad integrarsi con altre piattaforme che forniscono servizi di ML (come *Kubeflow*). Per esempio, un servizio di AI di base potrebbe essere ottenuto aggiungendo attributi, oltre a quelli già presenti nel modello dati, dedicati esclusivamente al processo di machine learning. Tutto questo è possibile grazie all'adozione di standard aperti, all'elasticità di utilizzo dei Data Model e dello standard NGSI-LD di FIWARE.

Bibliografia

[1] Ahle, Ulrich, e Juan Jose Hierro. «FIWARE for Data Spaces». *Designing Data Spaces : The Ecosystem Approach to Competitive Advantage*, a cura di Boris Otto et al., Springer International Publishing, 2022, pp. 395–417. *Springer Link*, https://doi.org/10.1007/978-3-030-93975-5_24.

[2] Morgan, Lorraine, e Patrick Finnegan. «Benefits and Drawbacks of Open Source Software: An Exploratory Study of Secondary Software Firms». *Open Source Development, Adoption and Innovation*, a cura di Joseph Feller et al., Springer US, 2007, pp. 307–12. *Springer Link*, https://doi.org/10.1007/978-0-387-72486-7_33.

[3] VanDerHorn, Eric, e Sankaran Mahadevan. «Digital Twin: Generalization, characterization and implementation». *Decision Support Systems*, vol. 145, giugno 2021, p. 113524. *ScienceDirect*, https://doi.org/10.1016/j.dss.2021.113524.

[4] Conde, Javier, Andrés Munoz-Arcentales, et al. «Modeling Digital Twin Data and Architecture: A Building Guide With FIWARE as Enabling Technology». *IEEE Internet Computing*, vol. 26, fasc. 3, maggio 2022, pp. 7–14. *IEEE Xplore*, https://doi.org/10.1109/MIC.2021.3056923.

[5] Conde, Javier, Andres Munoz-Arcentales, et al. «Applying digital twins for the management of information in turnaround event operations in commercial airports». *Advanced Engineering Informatics*, vol. 54, ottobre 2022, p. 101723. *ScienceDirect*, https://doi.org/10.1016/j.aei.2022.101723.

Sitografia

Press corner European Comission : https://ec.europa.eu/commission/presscorner/home/en

Decennio digitale europeo: obiettivi 2030: https://commission.europa.eu/strategy-and-policy/priorities-2019-2024/europe-fit-digital-age/europes-digital-decade-digital-targets-2030_it

Meditech: https://www.meditech4.com

FIWARE - Open APIs for Open Minds: https://www.fiware.org

FIWARE Academy: https://fiware-academy.readthedocs.io/en/latest

FIWARE Tour Guide. https://fiwaretourguide.readthedocs.io/en/latest

JSON: https://www.json.org/json-it.html

JSON-LD - JSON for Linking Data: https://json-ld.org

Data Models – FIWARE: https://www.fiware.org/data-models

About Node-RED: https://nodered.org/about

Cos'è MongoDB? : https://www.mongodb.com/it-it/company/what-is-mongodb.

Cos'è Grafana? : https://www.redhat.com/it/topics/data-services/what-is-grafana

Postman API Platform: https://www.postman.com

NGSI-LD Linked Data: https://documenter.getpostman.com/view/513743/SVYjSMgh

FIWARE – Smart Cities: https://www.fiware.org/about-us/smart-cities

Subscriptions NGSI-v2: https://fiware-tutorials.readthedocs.io/en/latest/subscriptions.html

Grieves and Vickers, the History of Digital Twins: https://diginomica.com/grieves-and-vickers-history-digital-twins