Controllo remoto per dispositivi IoT in realtà aumentata

Dipartimento di Ingegneria Enzo Ferrari
Corso di Laurea in Ingegneria Informatica

Candidato
Daniele Napolitano
Matricola 276285

Relatore
Prof. Nicola Bicocchi

Correlatore
Prof. Marco Picone

Anno Accademico 2021/2022

Controllo remoto per dispositivi IoT in realtà aumentata
Tesi di Laurea. Università di Modena e Reggio Emilia

LaTeX.

Email dell'autore: 276285@studenti.unimore.it

Ai miei genitori,

a Martina,

agli amici,

e a nonna Liliana.

Sommario

Nella presente tesi si introdurranno i concetti di Digital Twin, Realtà Aumentata e Internet of Things, che insieme hanno un ruolo importante nell'industria 4.0.
Verranno descritti sotto diversi punti di vista, e si presenterà la realizzazione di un caso di studio: un software per controllare dispositivi IoT nella rete locale, tramite un'interfaccia utente in realtà aumentata.
Verranno analizzati in dettaglio il funzionamento del protocollo CoAP e i metodi di motion tracking di ARCore.
Si discuterà infine delle possibli applicazioni, e sulle implicazioni di questi strumenti in diversi ambiti reali, presentando casi d'uso e valutando pro e contro di queste tecnologie.

Indice

Capitolo 1

Introduzione

1.1 Digital Twin

Con Digital Twin, che si traduce letteralmente come “Gemello Digitale”, si indica la rappresentazione virtuale di un qualsiasi sistema reale (ad esempio un dispositivo elettronico, una persona, un macchinario, o anche un processo) durante tutte le fasi del suo ciclo di vita, con lo scopo di facilitare l’analisi, lo sviluppo e il testing.
Un’infrastruttura digital twin è composta da tre componenti:

1. Una parte fisica.
2. Una parte digitale (che virtualizza l’ente fisico nella sua interezza).
3. Tutto ciò che collega la parte fisica a quella digitale (ad esempio sensori per il monitoraggio, attuatori per l’interazione, protocolli di comunicazione, ecc.).

L’idea è stata introdotta per la prima volta negli anni ‘60 dalla NASA durante il periodo delle missioni Apollo, nelle quali vennero create delle copie identiche delle astronavi, ma tenute sulla Terra, con lo scopo di analizzare e simulare con più precisione possibile il comportamento del gemello lunare in scenari e condizioni particolari. Col tempo e con l’evoluzione dei computer, questi gemelli fisici sono diventati virtuali, ed il concetto di digital twin è stato formalizzato nel 2010 dalla stessa agenzia spaziale.[10]

1.1.1 Vantaggi

Si potrebbero confondere i digital twin con il concetto di simulazione, in quanto entrambi sfruttano modelli digitali per rappresentare qualcosa di reale, ma la simulazione si concentra solo su un aspetto particolare del sistema, mentre il digital twin è una rappresentazione globale, e può essere messo in diretta relazione con il suo gemello reale (ciò non è vero per le simulazioni, in quanto sono rappresentazioni parziali). Il digital twin è poi in grado di effettuare diverse simulazioni sul suo modello, per studiare aspetti diversi.
Per le applicazioni più avanzate, allo stato dell'arte attuale, i digital twin vengono integrati con modelli di machine learning per fornire un supporto alle scelte decisionali, ispezionando autonomamente vaste quantità di dati (ad esempio consigliando di effettuare la manutenzione di un macchinario con largo anticipo, sulla base delle sue performance). [5]

1.1.2 Applicazioni e casi d'uso

I digital twin possono essere implementati in tantissimi scenari differenti. Sono spesso associati con il concetto di industria 4.0, e sempre più presenti in questo ambito, in quanto permettono di rappresentare globalmente processi complicati, composti da molteplici passaggi.
Sono utilizzati come strumento per monitorare processi industriali in tempo reale con l'aiuto di molti sensori, e prevedere in anticipo possibili problemi o malfunzionamenti. Sono in grado di ottimizzare i processi ed individuare possibili cause di inefficienza, fare analisi su scenari ipotetici (simulazioni), e rendono più semplice integrare i dati raccolti per altri software della stessa azienda.
Nelle prossime sezioni si introdurranno tecnologie relativamente recenti che hanno contribuito alla diffusione dei digital twin.

1.2 Realtà aumentata

La realtà aumentata (spesso semplicemente indicata come "AR") è un metodo visuale di rappresentare dati o oggetti, inserendoli nello spazio tramite la manipola-

zione del video in tempo reale dell'ambiente circostante.
In sostanza consiste nell'aggiungere ai frame delle immagini catturate circa 30 volte al secondo, degli elementi aggiuntivi (che possono essere oggetti 3D, testo, immagini, ecc.). Questi vengono integrati in modo coerente nell'ambiente, in modo da fornire l'illusione di essere realmente presenti in esso (ad esempio adattando la grandezza e la rotazione dell'oggetto per rispettare la prospettiva) facendoli sembrare fissi nell'ambiente, anche quando l'utente si sta muovendo in giro.
E' necessario che il dispositivo in uso abbia una certa comprensione dello spazio circostante, inteso come la sua posizione ed i suoi spostamenti nel tempo, la quantità di luce presente, ed altre caratteristiche. Sono quindi necessari diversi sensori come giroscopio, accelerometro, sensori di prossimità, o altri per implementare tecniche di motion e depth tracking. Dovendo manipolare decine di frames ogni secondo sulla base dei dati dei sensori, la realtà aumentata è abbastanza pesante dal punto di vista computazionale, per questo motivo si è diffuso solo di recente, ed utilizzato soprattutto su dispositivi mobile, che anno dopo anno diventano sempre più performanti, e sono spesso dotati di tutti i sensori sopra citati.
Sono in sviluppo da diverse aziende i cosiddetti *mixed reality glasses*: visori in grado di proiettare gli elementi virtuali sulle lenti o direttamente nella retina degli occhi, unendo realtà virtuale e realtà aumentata per fornire un'esperienza molto più immersiva.

1.2.1 Digital Twin e AR

La realtà aumentata aiuta a mostrare elementi virtuali in modo molto intuitivo ed accessibile, più efficacemente che guardando la stessa cosa su uno schermo bidimensionale. Questo vale anche per i dati, in quanto si è in grado di gestire un'elevata mole di informazioni provenienti da sensori, visualizzandoli in modo intuitivo, aiutando la loro interpretazione.
Per questo motivo possono essere usati anche come strumento per fornire nuove modalità di istruzione (sia scolastiche che in ambito lavorativo) molto più efficienti rispetto al modo tradizionale.

1.3 Internet of Things

Con Internet of Things (IoT) si intende un sistema di dispositivi dotati di sensori che comunicano tra loro (Machine to Machine communication, o “M2M”), e che non hanno bisogno di interazioni umane, in modo che questi oggetti (ad esempio elettrodomestici, orologi, o anche lampioni, semafori, e così via) oltre a comunicare con gli utenti umani possano comunicare tra loro scambiandosi continuamente dati, costituendo appunto un "internet delle cose".
In genere i dispositivi in questione non sono dotati di capacità computazionale elevata, anzi spesso dispongono di un microprocessore embedded capace solo di leggere dati dai sensori e spedirli sulla rete, e sono caratterizzati da un consumo energetico molto basso.
Il concetto è stato discusso per la prima volta agli inizi degli anni ’80, quando un distributore di lattine di Coca-Cola modificata dell’università di Carnegie Mellon venne collegato ad Arpanet, ed era in grado di comunicare quante lattine contenesse, e la temperatura al suo interno.

1.3.1 Applicazioni

Le applicazioni di questa tecnologia spaziano in una quantità molto vasta di campi totalmente diversi. Per citarne una, l’IoT ha un ruolo fondamentale nell’industria 4.0, in quanto la capacità di monitorare tanti dispositivi è utile per controllare minuziosamente i processi industriali. Nel caso della domotica, l’IoT è una parte chiave per la cosiddetta home automation, permettendo di creare un sistema interconnesso di luci, sistemi di riscaldamento, condizionatori, allarmi, ecc., il tutto controllabile da remoto, con la possibilità di sfruttare automazioni.
Un altro ambito interessante è quello delle smart cities, permettendo di monitorare una vastissima quantità di dispositivi sparsi per la città (ad esempio semafori, parcheggi, luci, trasporti pubblici, e molti altri), con lo scopo di rendere più efficiente la viabilità, garantire servizi e supportare la gestione energetica ed ambientale.

1.3.2 Digital Twin e IoT

Un prerequisito dei digital twin è quello di monitorare diversi aspetti tramite sensori, pertanto si trova quasi sempre associato a dispositivi IoT: nel caso della domotica si può considerare un digital twin della casa, composto da tutti i dispositivi presenti in essa, o per la smart city è possibile produrre digital twin di intere città riproducendo la mappa della zona, integrando i dati ottenuti da tutti i dispositivi IoT sparsi in essa.

1.4 Caso di studio

Per realizzare nella pratica quanto discusso in questo capitolo, è stato realizzato il seguente progetto dimostrativo: applicazione Android in realtà aumentata, che funziona da controllore per dispositivi IoT sulla rete locale, mostrandone i dati dei sensori ed eventuali pulsanti/slider per l'interazione in tempo reale. Tutto questo in realtà aumentata, cioè sovrapponendo questi elementi virtuali su un'immagine particolare associata ad un dispositivo IoT.

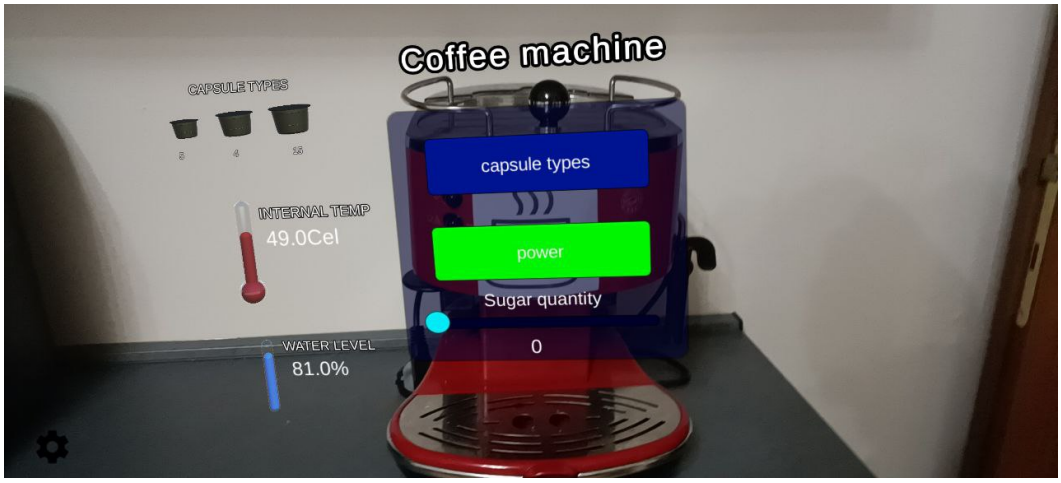

Figura 1.1. *Screenshot del client in funzione*

Capitolo 2

Tecnologie Usate

Nel presente capitolo si discuterà dei vari programmi, tecnologie e protocolli utilizzati per la realizzazione del progetto, trattando quindi ciò che sarà necessario per comprenderne il funzionamento.
Trattandosi di un applicativo distribuito, sono stati utilizzati più IDE e linguaggi differenti.

2.1 Unity

Per realizzare l'applicazione del client per lo smartphone, è stato utilizzato Unity, un motore grafico per lo sviluppo di videogiochi e contenuti interattivi in 3D o 2D. E' basato sul linguaggio C#, pertanto è ben integrato con Visual Studio, ma il software di Unity ha un'interfaccia utente intuitiva, offrendo strumenti in grado di gestire con facilità e con meno codice possibile gli aspetti grafici del progetto (esempio animazioni, shader, texture, manipolazione di modelli 3d, ecc.). Questo porta però anche ad alcune limitazioni (come l'impossibilità di usare costruttori per le classi, la scarsa compatibilità con librerie esterne, la difficoltà nello sfruttare il multithreading).
La popolarità di questo software ha reso disponibili moltissime guide online e plugin aggiuntivi, che hanno aiutato molto la fase di sviluppo, garantendo un risultato visivo discreto pur non disponendo di abilità grafiche e di design particolari.
Oltre a questi motivi, è stato scelto per la sua natura cross-platform, in quanto

è possibile eseguire il deploy dei progetti su un numero elevatissimo di dispositivi diversi, tra i quali smartphone (sia Android che iOS), console di gioco, e i visori per Realtà Virtuale e Realtà Aumentata più diffusi.
È stato utilizzato il programma Blender per realizzare semplici modelli 3D.

2.1.1 ARFoundation

AR Foundation è un framework multipiattaforma di Unity, che consente di sviluppare software di Realtà Aumentata compatibile con varie SDK di diversi dispositivi, come ARKit di iOS, ARCore di Android (supportato anche da iOS), o XR Microsoft per HoloLens. Fornisce un'interfaccia comune agli sviluppatori per realizzare software AR, ma senza effettivamente implementare nessuna feature a riguardo, affidandosi completamente alle SDK.
E' molto utile in quanto, in questo modo, è possibile fare il deploy su tutta questa serie di dispositivi compatibili senza apportare modifiche al codice.

2.2 ARCore

Il progetto in questione è stato pensato per soli dispositivi Android, per motivi che verranno spiegati di seguito in questo capitolo, difatti viene utilizzato solo ARCore: la piattaforma di Google per sviluppare software AR mediante apposite API proprietarie.
Nella presente sezione verranno riportate in modo elementare alcuni metodi messi in atto da ARCore per il suo funzionamento.

2.2.1 Motion Tracking

Con questo termine si indicano le tecniche per rilevare il cambiamento di posizione di un oggetto sulla base dei dati registrati dai sensori.
ARCore prende spunto dal funzionamento del cervello, combinando i dati di:

- Telecamera (come gli occhi), per estrarre **feature points** con tecniche di computer vision, ossia trovare i punti dove c'è un elevato contrasto visivo, che risultano più facili da tracciare.

- Sensori **IMU (Inertial Measurement Unit)**, ossia giroscopio e accelerometro, che simulano l'apparato vestibolare presente nelle orecchie e fornisce il senso di equilibrio e di accelerazione, permettendo nel caso di ARCore di tener traccia più accuratamente degli spostamenti.

I dati combinati vengono forniti ad un modello di machine learning che migliora il tracciamento del dispositivo quando i dati della telecamera non sono affidabili (esempio se la luce è scarsa o se la lente viene coperta per qualche secondo).
Infine quello che si ottiene è una stima dell'orientamento del dispositivo, e dei suoi spostamenti nel tempo.

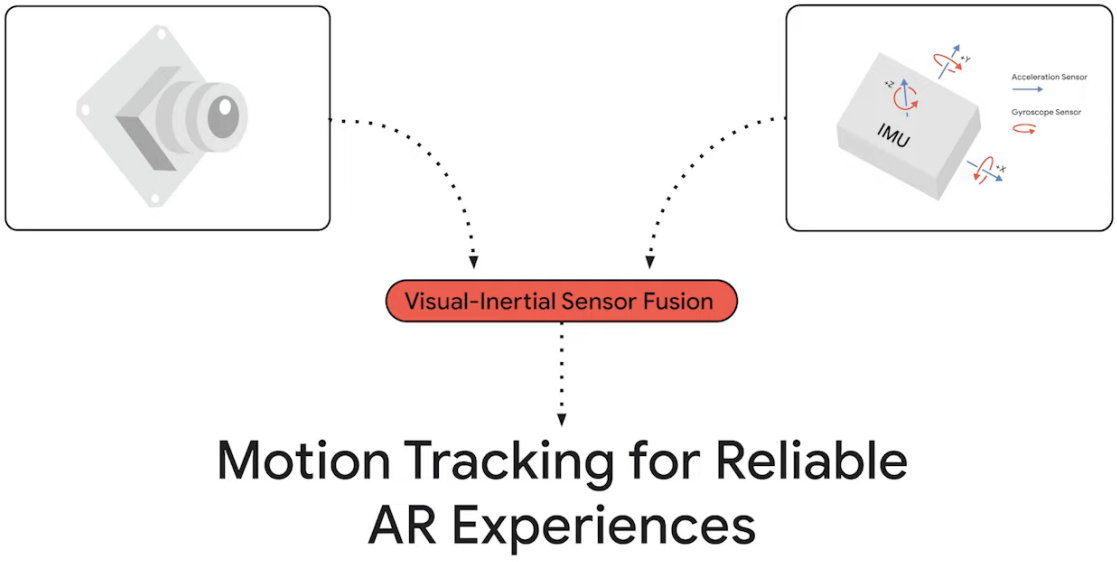

Figura 2.1. *Come gli input del dispositivo vengono usati per calcolare il motion tracking in ARCore[4]*

Per orientarsi nello spazio, il dispositivo deve risolvere un problema computazionale noto come **SLAM (Simultaneous Localization and Mapping)**, che, come suggerisce il nome, consiste nello svolgere queste due operazioni in contemporanea:

1. Creare una mappa virtuale dello spazio circostante.
2. Stimare la posizione del dispositivo all'interno di tale mappa.

È un processo complesso, ma che viene utilizzato in diversi campi dell'informatica, ad esempio per la guida autonoma, la robotica, la realtà virtuale e quella aumentata.

Matematicamente, presa una serie di intervalli di tempo discreti t, e date le serie temporali di controlli di input $U_{0:t}$ (ossia i dati dei sensori), le posizioni precedenti del dispositivo $X_{0:t}$, l'insieme di tutte le osservazioni dei feature points nello spazio $Z_{0:t}$, bisogna trovare la posizione attuale x_t e una mappa dell'ambiente m. Tutte queste entità sono probabilistiche, in quanto le misurazioni non saranno esatte. Si deve calcolare la seguente probabilità condizionata:

$$P(x_t, m|Z_{0:t}, U_{0:t}, x_0)$$

Si noti che la probabilità viene calcolata tenendo in considerazione tutte le misurazioni precedenti, dal tempo iniziale $t = 0$ fino all'istante attuale t.[2]
Esistono diversi algoritmi che risolvono il problema, in particolare ARCore utilizza il **MAP (Maximum a Posteriori Probability)**, il quale usa la regola di Bayes per stimare puntualmente la posizione più probabile in cui si trova il dispositivo.

2.2.2 Depth API

ARCore fornisce una API apposita per la stima della profondità, stimando la distanza degli oggetti ripresi dalla telecamera fino a 65 metri di distanza.
Questa stima avviene usando algoritmi di "depth from motion", che si basano esclusivamente sulle immagini della telecamera e lo spostamento del dispositivo: prendendo più frame dello stesso oggetto visto da angolazioni diverse e comparandoli tra loro, riesce ad associare ad ogni pixel una distanza approssimativa, creando una depth image dell'ambiente (esempio in Figura 2.2).
Sostanzialmente si crea un modello 3D approssimativo dello spazio, ciò aiuta ulteriormente il dispositivo ad orientarsi, e permette di occludere elementi virtuali dietro ad ostacoli reali (come pareti o altre superfici grosse), fornendo un senso di immersività ancora più realistica.

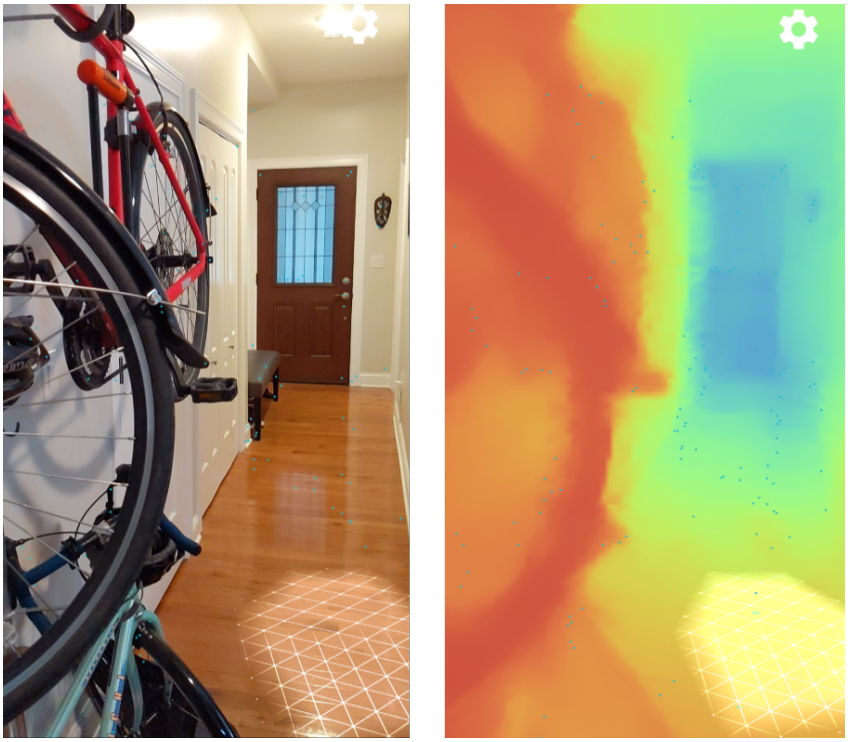

Figura 2.2. *Esempio di depth image ottenuto dalla depth API di ARCore[4]*

2.2.3 Augmented images

ARCore permette di effettuare il riconoscimento di immagini 2D (anche in movimento), su cui istanziare degli oggetti 3D virtuali. Questa funzionalità viene denominata “Augmented Images”.

Viene ispezionata l’immagine in scala di grigio per trovare feature points, salvando i risultati in un database (nel caso di ARFoundation questo viene denominato "image library"). In fase di esecuzione, l’applicazione si mette in cerca delle caratteristiche sugli elementi piani che vengono inquadrati ad ogni frame. Quando viene correttamente riconosciuta un’immagine dell’image library, si applicano le tecniche di AR descritte nella sezione precedente, per stimare la posizione dell’immagine nello spazio e le sue dimensioni reali, in modo da poter inserire gli elementi virtuali in sovrapposizione con coerenza.

Le immagini usate devono rispettare alcuni requisiti per permettere l’estrazione dei feature points: devono avere elementi con alto contrasto, e non devono contenere pattern visivi ripetuti (altrimenti i feature points si confondono e si ripetono). Pos-

sono essere inserite fino a 1000 immagini in una image library, ma si può tenere traccia solo di 25 immagini contemporaneamente (e queste possono anche essere aggiunte o rimosse a runtime).

2.3 CoAP

CoAP (Constrained Application Protocol), è un protocollo di rete a livello applicativo, definito ufficialmente nel 2014 con il documento RFC 7252. [7]
E' stato pensato, come evidenziato dal nome, per dispositivi constrained, ossia con hardware limitato. Ideale per comunicazioni in ambito Internet of Things (IoT), Machine to Machine (M2M), e per dispositivi con poca memoria RAM e bassa capacità di calcolo. Essendo un protocollo di tipo REST (più precisamente CoRE, ossia Constrained REST), è ben integrato con HTTP, difatti è stato sviluppato in modo da facilmente convertire le richieste HTTP in CoAP (tramite cosiddetti "CoAP Proxy"), anche se presenta alcune differenze. La più immediata, è che CoAP si basa su UDP (anche se esistono modi per usarlo su TCP), questo per ridurre notevolmente l'overhead ed evitare il carico computazionale necessario per mantenere una connessione TCP, ma introduce dei metodi per rendere la connessione affidabile.

2.3.1 Header

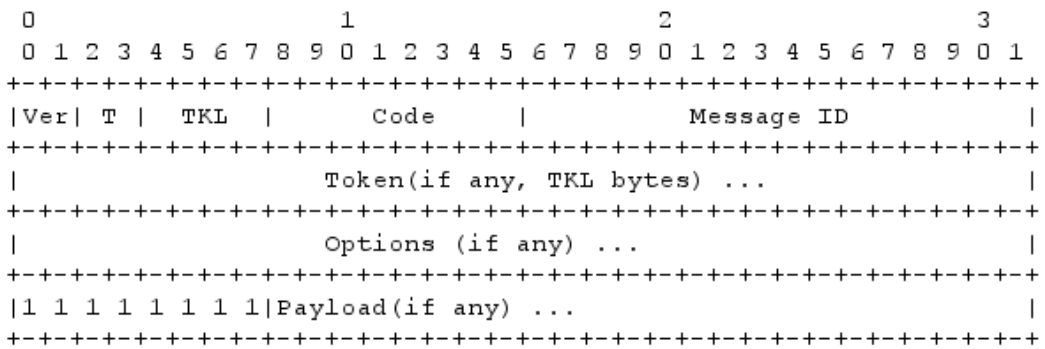

Figura 2.3. *Header del protocollo CoAP*

Il secondo campo "Type" (indicato come "T" in Figura 2.3), è lungo 2 bit e si occupa di gestire l'affidabilità del protocollo. Questa è importante in quanto CoAP viene usato su dispositivi "lossy", ossia che tendono a perdere molti pacchetti a

causa della scarsa potenza di calcolo.
Token viene usato per correlare tra loro request e response (la lunghezza è data dal campo TKL).
Il campo Code è analogo a quello di HTTP, serve cioè ad indicare tramite cifre ad 8 bit informazioni sul messaggio (esempio 0 se è una request, 2 per successful response, 4 per errori del client, 5 per server error, ecc.).
Il campo Message ID serve ad identificare univocamente il pacchetto, quindi a scartare eventuali duplicati.
Options e Payload sono opzionali.

2.3.2 Discovery

I sistemi CoRE (Constrained Restful Environments), standardizzati dal RFC 6690[8] e utilizzati come anticipato prima per la comunicazione machine-to-machine, hanno in comune la possibilità di effettuare Resource Discovery tramite l'uri standard */.well-known/core*.
Utilizzando il CoRE Link format (un'estensione del Web Linking dell'RFC 5988, usato anche per HTTP), ogni URI di una risorsa è associato a una serie di attributi standard.
In pratica permette di scoprire (in inglese appunto "discover"), conoscendo solo l'IP del server CoAP e la sua porta, tutte le risorse messe a disposizione da quest'ultimo, i relativi URI per raggiungerle, e per ognuno una serie di attributi che riportano informazioni aggiuntive. Questa possibilità permette di sviluppare una comunicazione automatica tra più dispositivi di tipo CoRE.
Gli attributi standard sono tutti quelli presenti per il Web Linking, con l'aggiunta di alcuni specifici per CoRE; alcuni tra i più importanti sono:

- **Interface (if):** per indicare il tipo di dispositivo, usa valori standard (*core.a* per attuatori, *core.s* per sensori).
- **Resource type (rt):** indica più nello specifico il tipo della risorsa, e quindi anche come interfacciarsi ad essa, ma non richiede valori standard (ad esempio nell'app si riferisce a "btn" per indicare che si tratta di un bottone).

- **Title (title):** un nome simbolico per il dispositivo.
- **Name (n):** Il nome del dispositivo, destinato ad altre macchine (quindi spesso cifre e caratteri poco significativi per gli umani).
- **Content type (ct):** fornisce delle indicazioni per il formato della risposta (es. text/plain), sotto forma di codice numerico standard.

Si noti che ne esistono anche altri più o meno importanti, ma che non sono stati ritenuti rilevanti in quanto non utilizzati nel progetto. Si riporta anche un esempio di risposta completa di discovery, presa direttamente dal software sviluppato:

```
</slider>;ct="1100";if="core.a";rt="slider";title="slider",
</temperature>;ct="1100";if="core.s";rt="temp";title="internal temp"
```

2.3.3 SenMl

Sta per Sensor Measurement List, definito nel RFC 8428[6] per fornire un modo efficace e leggero di rappresentare le misurazioni di sensori. Viene spesso utilizzato assieme a CoAP, in quanto è stato studiato per dispositivi dotati di processori con capacità computazionale limitata, pur essendo capace di immagazzinare una grande quantità di dati provenienti da sensori in modo efficiente. SenML permette di allegare metadati auto-descrittivi che non sono riportati dalla discovery.
I dati sono strutturati in un singolo array, che contengono una serie di record SenML. Ognuno di questi contiene i campi che descrivono il sensore e i dati della misurazione; l'array può essere successivamente serializzato in JSON, XML, CBOR o EXI.
I campi più importanti ed usati per l'applicazione sono:

- **Name (n):** per indicare di che dato si tratta.
- **Value (v):** contiene il valore misurato (v è per dati generici, vb indica valori booleani e vs quelli in stringhe).
- **Unit (u):** per specificare l'unità di misura.

Di seguito è riportato l'esempio di un array composto di un solo record SenML serializzato in JSON, che utilizza i campi sopra descritti:

```
[
    {"n":"urn:dev:ow:10e207","u":"Cel","v":23.1}
]
```

2.4 Android Studio

Per realizzare il server CoAP è stata utilizzata la libreria Java Californium, usando l'IDE IntelliJ IDEA. Non essendoci librerie analoghe scritte in C# e compatibili con Unity, anche le richieste lato client (sullo smartphone) vengono effettuate con la libreria Californium, sempre in Java. Questo è stato reso possibile dai *native plugin*[9] di Unity: una funzionalità che permette di eseguire codice da librerie pre-assemblate, e richiamare metodi o funzioni direttamente da essi sul codice in Unity.
Per realizzare queste librerie, è stato usato Android Studio per creare degli archivi in formato .ARR con classi statiche Java che possono eseguire chiamate CoAP di diversi tipi.
I metodi statici possono essere chiamati dal codice di Unity tramite metodi particolari, e allo stesso modo il plugin potrebbe chiamare dei metodi di Unity, anche se in questa direzione il processo sarebbe più complicato, e non è stato implementato nel progetto.
Nel prossimo capitolo (Listing 3.4 e 3.5) verranno mostrate anche parti di codice.

Capitolo 3

Caso di studio

Si presenteranno di seguito le scelte progettuali e le tecniche di ingegneria del software messe in atto per la realizzazione del software (discusso nella sezione 1.4), mostrando anche diagrammi UML per integrare le spiegazioni.
L'intero progetto si compone di tre parti (e attori) principali: il client sullo smartphone, il server HTTP e i vari server CoAP presenti sui dispositivi IoT. Nel capitolo verrà trattato con dettaglio maggiore il client, in quanto esso accetta qualsiasi server CoAP, indipendentemente dalla sua implementazione.

3.1 Requisiti

Come requisito funzionale, è essenziale che l'interfaccia utente sfrutti al meglio la natura tridimensionale della realtà aumentata, risultando piacevole da guardare ed allo stesso tempo intuitiva; questo punto è infatti il fattore di successo dell'applicativo, che lo renderebbe preferibile rispetto a un'app analoga in 2D.

Ci sono poi altri requisiti non funzionali importanti che sono stati presi in considerazione durante lo sviluppo:

- **Performance:** le operazioni di ARCore che vengono applicate ad ogni frame richiedono molte risorse computazionali, quindi il resto dell'applicazione deve essere progettata per essere il più efficiente possibile da questo punto di vista.

- **Generalità:** Il client deve poter supportare qualsiasi server CoAP, pertanto deve poter essere in grado di visualizzare qualsiasi tipo di dato e interagire con qualsiasi attuatore.
- **Modificabilità, manutenibilità e scalabilità:** Nonostante il punto precedente, deve essere possibile aggiungere interfacce specifiche per alcuni dispositivi, per sfruttare al meglio la realtà aumentata; questo processo deve essere il più semplice possibile.

3.1.1 Use Case Diagram e architettura generale

In Figura 3.1 è mostrato lo Use Case Diagram che riporta le azioni base compiute dai due attori.

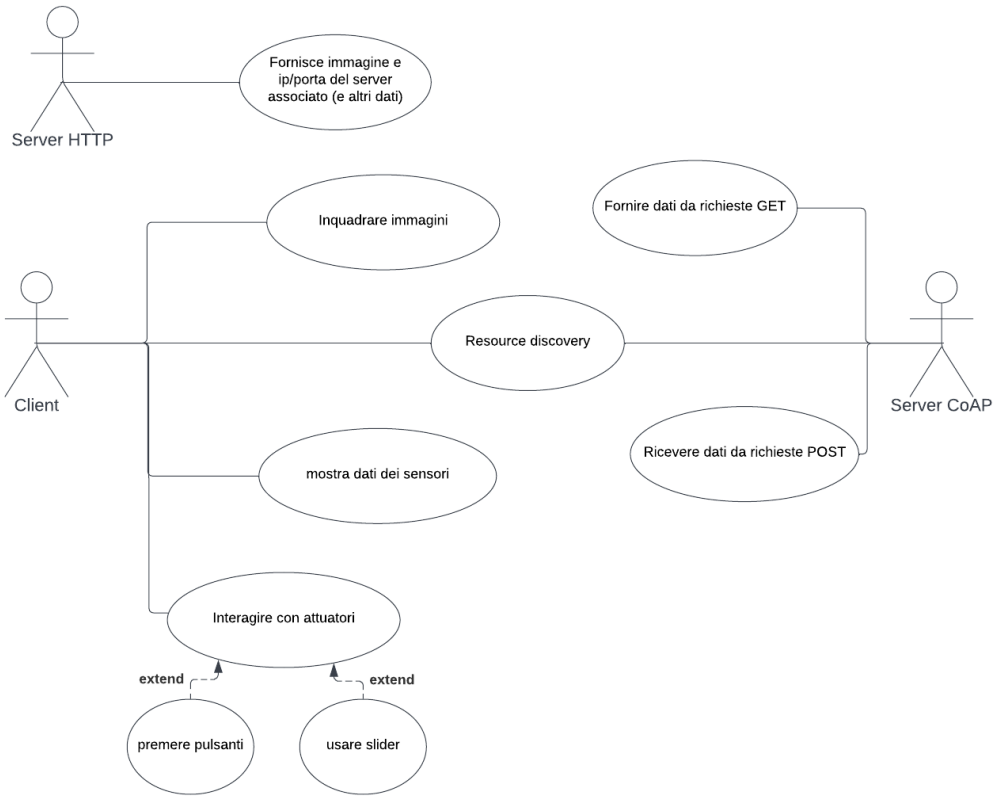

Figura 3.1. *Use case diagram*

In Figura 3.2 è riportato uno schema che mostra le procedure svolte in ordine numerato dal sistema client/server durante tutta l'esecuzione del client.

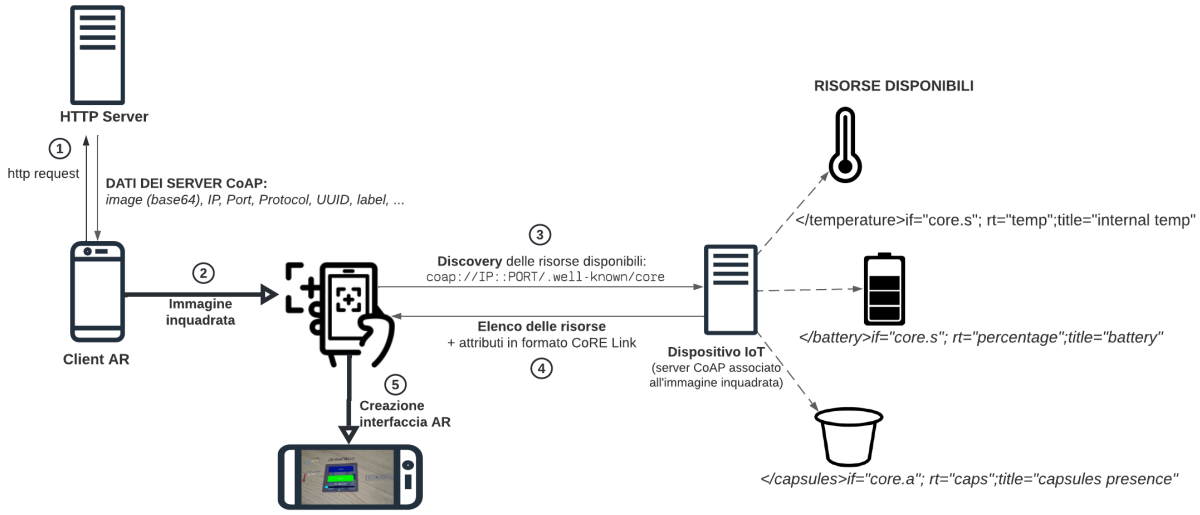

Figura 3.2. *Architettura del sistema client/server*

Verranno qui brevemente spiegati i passaggi, che verranno approfonditi più nel dettaglio nelle prossime sezioni.

1. Al momento del primo avvio, il client contatta l'API del server HTTP, il quale gli risponde con le informazioni sui server CoAP presenti sulla rete locale.

2. L'utente che usa l'applicazione AR (il client), inquadra attraverso la fotocamera dello smartphone una delle immagini comunicate dal server HTTP.

3. Il client effettua una richiesta Discovery al server CoAP associato a tale immagine.

4. Il server risponde con l'elenco delle risorse disponibili in formato Core Link (uri e attributi).

5. Sulla base di queste informazioni, il client crea un'interfaccia utente che viene sovrapposta sull'immagine del server (augmented image).

3.1.2 Interfaccia utente e nomenclature

Con riferimento ai numeri in Figura 3.3, verranno mostrati di seguito gli elementi principali dell'interfaccia utente e una breve descrizione per ognuno di essi

(queste nomenclature verranno usate nel corso del capitolo anche per indicare le corrispondenti classi).

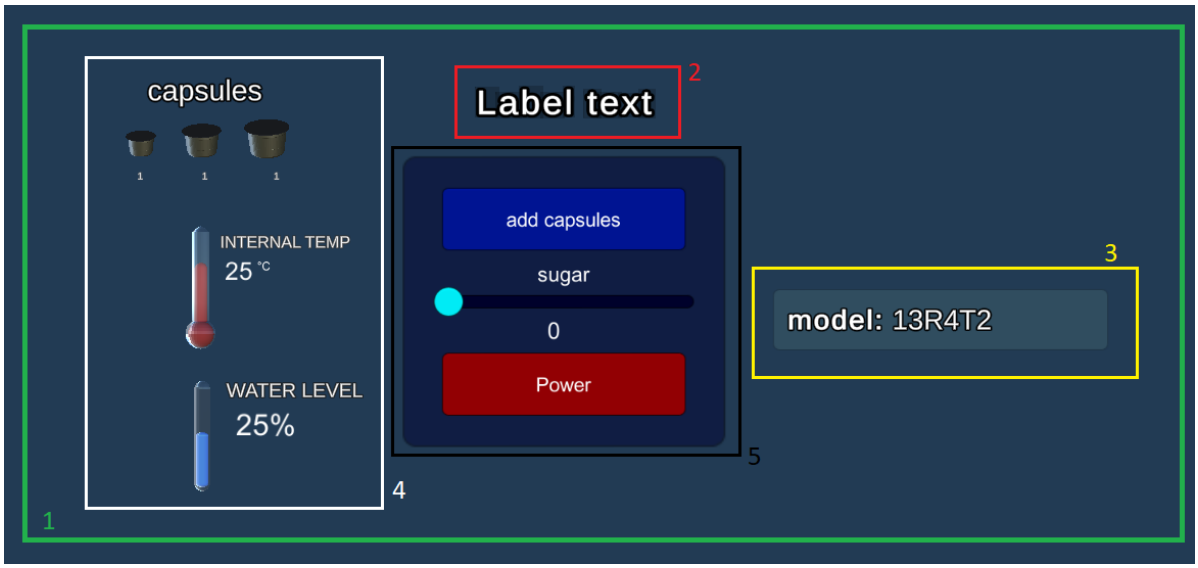

Figura 3.3. *Interfaccia relativa al server di una macchinetta del caffè, con elementi numerati*

1. **Canvas:** L'intera interfaccia, relativa ad un dispositivo specifico (ossia correlato all'immagine appena inquadrata).

2. **Label:** il nome del digital twin.

3. **Monitor:** sezione in cui vengono mostrati diversi tipi di dati (ottenuti mediante richieste GET al server CoAP), ogni categoria viene preceduta da un titolo in grassetto. Ad ogni monitor corrisponde una risorsa di cui non si conosce il campo if (è la vista default per sensori).

4. **Sensori 3D**: Una sezione riservata a dati visualizzabili in modi alternativi, ossia non tramite semplice testo. Pensata per sfruttare a pieno la natura AR dell'applicazione, ad esempio visualizzando un termometro per indicare un sensore di temperatura.

5. **Attuatori**: sezione in cui sono presenti tutti gli elementi con i quali è possibile interagire (es. bottoni o slider). Ognuno ha un label utile a comprenderne il suo scopo.

3.2 Class Diagram

In Figura 3.4 è riportato il class diagram complessivo del client, suddividendo le classi in package.

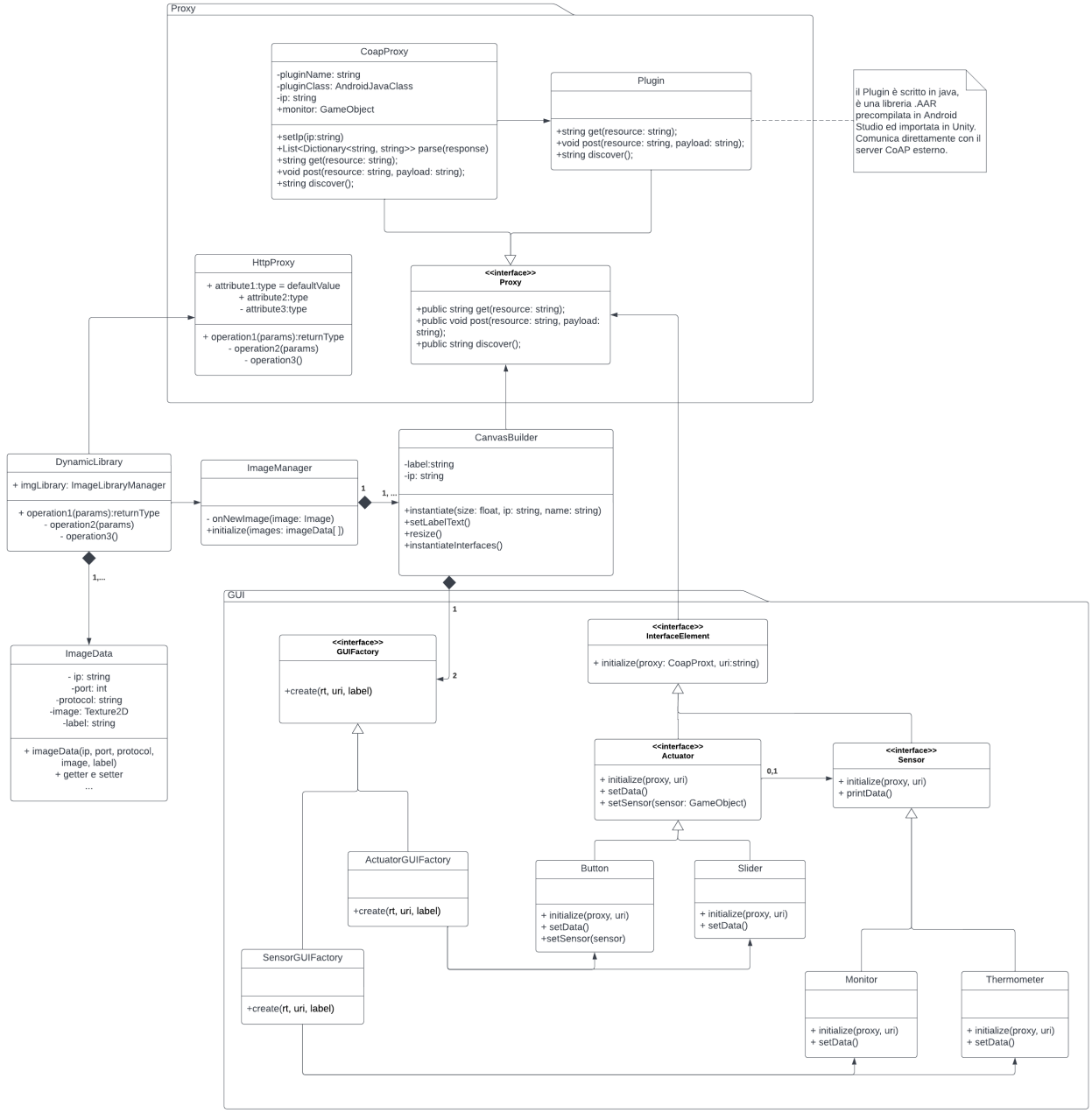

Figura 3.4. *Class diagram complessivo del client*

Si precisa che per le classi che implementano l'interfaccia *MonoBehaviour* della libreria di Unity (ossia la quasi totalità delle classi presenti nel diagramma) non si

può definire il costruttore, e sono istanziate al momento dell'avvio. Per inizializza il metodo *initialize([argomenti])*, usandolo come se fosse un costruttore per inizializzare attributi e chiamare altri metodi.

3.2.1 Primo avvio e comunicazione con server HTTP

Nel momento in cui viene fatta partire l'applicazione, la classe DinamicLibrary chiama l'API del server HTTP all'uri *http://<ip>:7070/api/iot/inventory/device*, che fornirà una serie di informazioni essenziali su tutti i dispositivi IoT presenti sulla rete locale.
Queste vengono salvate in un array di oggetti *ImageData*, i quali attributi più rilevanti sono:

- **uuid**: numero identificativo del dispositivo.
- **displayName**: nome simbolico del dispositivo, sarebbe il Label in figura 3.3.
- **ip, porta e protocollo**: informazioni che permettono di contattare il server (si presuppone che il protocollo sia CoAP, ma si potrebbe sviluppare prendendo in considerazione qualsiasi protocollo RESTful).
- **image**: l'immagine riportata per comodità in formato base64, ossia come stringa (viene poi convertita in formato Texture2D, ed aggiunta al database di immagini di AR Foundation).

Listing 3.1. Codice Java del server HTTP per l'aggiunta di dispositivi

```
1     private void addDeviceCoffee(AppConfig appConfig){
2         DeviceDescriptor deviceDescriptor= new DeviceDescriptor();
3         deviceDescriptor.setUuid("device00001");
4         deviceDescriptor.setIp("192.168.0.225");
5         deviceDescriptor.setPort(5683);
6         deviceDescriptor.setProtocol("coap");
7         deviceDescriptor.setDisplayName("Coffee machine");
8         File img = new File("..\\services\\images\\image.jpg");
9         deviceDescriptor.setImage(base64Converter(img));
10
11        appConfig.getDeviceManager().createNewDevice(
              deviceDescriptor);
12    }
```

3.2.2 GUI

L'applicazione in questione non è altro che un'interfaccia utente adattata per funzionare in realtà aumentata: la struttura base del codice è fondamentalmente la stessa di un'interfaccia classica in 2D.
Ci sono due interfacce principali: Actuator e Sensor che distinguono interfacce concrete di due tipi. Sono molto simili, ma siccome svolgono compiti diversi, la differenza sta in due metodi particolari: in Sensor è presente *showData()* che esegue una GET e stampa i dati, mentre Actuator si distingue per i metodi *setValue()*, che si occupa di eseguire richieste POST e *setSensor(Gameobject sensor)*, che invece permette di associare una sottoclasse di Sensor a un Actuator (per aggiornare automaticamente i dati mostrati quando si interagisce con un attuatore). Entrambe le interfacce sono figli di InterfaceElement, la quale impone gli argomenti da passare nel metodo *initialize*, tra i quali deve essere presente la reference al CoapProxy specifico di quel canvas.

3.2.3 Creazione Canvas

Le classi ImageManager e CanvasBuilder sono responsabili per creazione del canvas del dispositivo associato all'immagine appena riconosciuta.
La classe ImageManager eredita dei metodi listener che si attivano quando viene riconosciuta una nuova immagine per la prima volta. Le procedure sono mostrate nel diagramma in Figura 3.5.

Il processo di discovery e di creazione del canvas verrà spiegato con maggior dettaglio nella sezione 3.4 e con il sequence diagram in Figura 3.7.

3.3 Design Patterns

Per adottare scelte di design coerenti, e per rendere modulare il più possibile modulare il software rispettando i principi SOLID della programmazione ad oggetti, si è fatto affidamento su alcuni design patterns noti, specificati nel libro *"Design Patterns: Elements of Reusable Object-Oriented Software"* [3].
Per rispettare i requisiti specificati in sezione 3.1, la struttura del codice deve

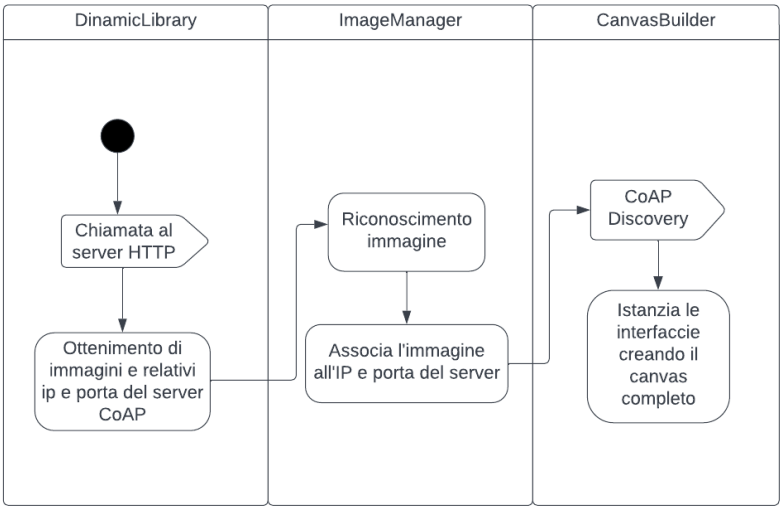

Figura 3.5. *Activity Diagram per il processo di riconoscimento immagine e creazione canvas (supponendo che il server sia raggiungibile)*

garantire che esso sia facile da estendere (fare in modo di dover modificare il minor numero di classi possibili), e ridurre gli sprechi di risorse computazionali.
L'utilizzo dei design pattern *Proxy* ed *Abstract Factory* è una delle possibili soluzioni per facilitare questo processo.

3.3.1 Proxy

Ogni tipo di interfaccia deve poter comunicare con il server (ognuna in un modo differente: ad esempio le interfacce che estendono l'interfaccia Sensor fanno solo richieste GET, mentre quelle Actuator possono in alcuni casi eseguire sia POST che GET). Per inviare richieste CoAP però, bisogna fare prima richiesta al plugin nativo in Java, un processo computazionalmente pesante per Unity. Sarebbe quindi molto inefficiente istanziare una classe plugin per ogni interfaccia (ad esempio nel caso in cui ci siano una decina di queste contemporaneamente le performance ne risentirebbero molto).
La soluzione potrebbe essere quella di istanziare solo una volta il plugin in una classe cha faccia da intermediario tra il plugin e tutte le interfacce (o ingenerale qualsiasi classe che abbia bisogno di comunicare con il server), il pattern più adatto sarebbe proprio il Proxy pattern.

La struttura delle classi è visibile in Figura 3.4 nel modulo Proxy.
Oltre al vantaggio in termini di ottimizzazione delle prestazioni, questa soluzione ha diverse implicazioni anche dal punto di vista della modificabilità e manutenibilità del codice:

- Trasforma ed adatta i dati forniti dal plugin (e quindi dal server), in modo che chi ne faccia richiesta ottenga direttamente i dati di cui ha bisogno (evitando anche ripetizioni di codice per la gestione dei dati, che è sempre uguale).
- Il proxy e il plugin sono completamente trasparenti all'interfaccia che ne richiede l'uso (e non vanno modificati in caso di modifica o aggiunta di interffacce).

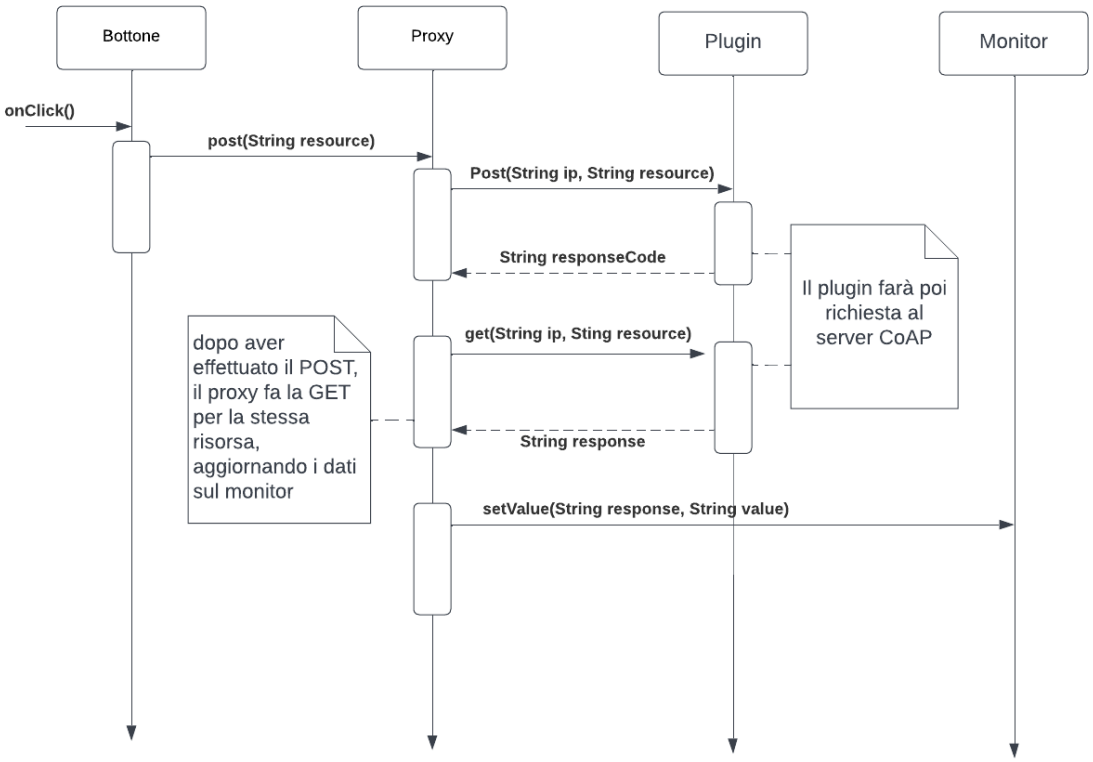

Figura 3.6. *Sequence Diagram che rappresenta le operazioni svolte in caso di richiesta POST da un bottone, passando dal Proxy*

3.3.2 Factory

Osservando la gerarchia delle interfacce nel class diagram (Figura 3.4), le classi che implementano Actuator devono essere inizializzate diversamente da quelle che implementano Sensor, e questo significa che anche il processo di istanziazione deve essere diverso. A tale scopo, è stato utilizzato il design pattern Factory per differenziare i processi, apportando i seguenti vantaggi:

- Quando si aggiungono nuove interfacce particolari, non è necessario modificare CanvasBuilder, ma basta solo aggiungere un singolo controllo if nel Factory concreto che corrisponde al tipo di interfaccia in questione.
- La classe CanvasBuilder non ha reference verso alcuna interfaccia, sono solo nelle Factory concrete.

In questo modo viene rispettato il principio open-closed.
Si riporta di seguito una parte di codice della classe ActuatorFactory per evidenziare la facilità col quale è possibile aggiungere nuovi attuatori (non è differente concettualmente dalla sua controparte per i sensori, anche se il codice è leggermente differente).

Listing 3.2. Metodo di istanziazione di ActuatorFactory

```
1  public void instantiateActuator(string type, string uri, string
       label)
2  {
3    GameObject newActuator;
4    if (type == "button")
5      newActuator = Instantiate(button, thisPosition, this.
           gameObject.transform.rotation);
6    else if (type == "slider")
7      newActuator = Instantiate(slider, thisPosition, this.
           gameObject.transform.rotation);
8      //per aggiungere interfacce si inserisce qui un nuovo else if
9    else
10     return;
11   newActuator.transform.localScale = new Vector3(0.5f, 0.5f, 1) /
         100;
12   newActuator.transform.parent = actuatorBackground.gameObject.
         transform;
13   newActuator.GetComponent<Actuator>().initialize(coapProxy, uri,
         label)
14 }
```

3.3.3 Analisi della stablità

Facendo riferimento al class diagram (Fig. 3.4), è evidente che le classi più stabili sono le interfacce Proxy e GUIFactory, in quanto attraggono gran parte delle dipendenze del grafico, ed essendo interfacce hanno il grado di astrazione più alto. Le classi CanvasBuilder e DinamicLybrary sono quelle più instabili, perchè sono come il "main" del client, in quanto si occupano dell'istanziazione del canvas e dei suoi componenti interni.

3.4 Discovery

La parte più critica dell'intero software è la creazione del canvas, in quanto il client non sa a priori come è strutturato il server, queste informazioni vengono scambiate appena dopo il riconoscimento dell'immagine, tramite una richiesta all'uri */.well-known/core*, al quale il server risponderà à con una lista di tutte le sue risorse disponibili (e per ognuna una serie di attributi standard per specificare come accedere e che tipo di interfaccia si tratti ad esempio), il tutto in formato CoRE link, come anticipato nel capitolo 2.
Il proxy trasforma la stringa della discovery response in una *List< Dictionary<string,string> >* in modo da essere facilmente letta ed iterata dal CanvasBuilder. Tutto questo processo passa attraverso molte classi, è stato riportato nel sequence diagram in Figura 3.7.

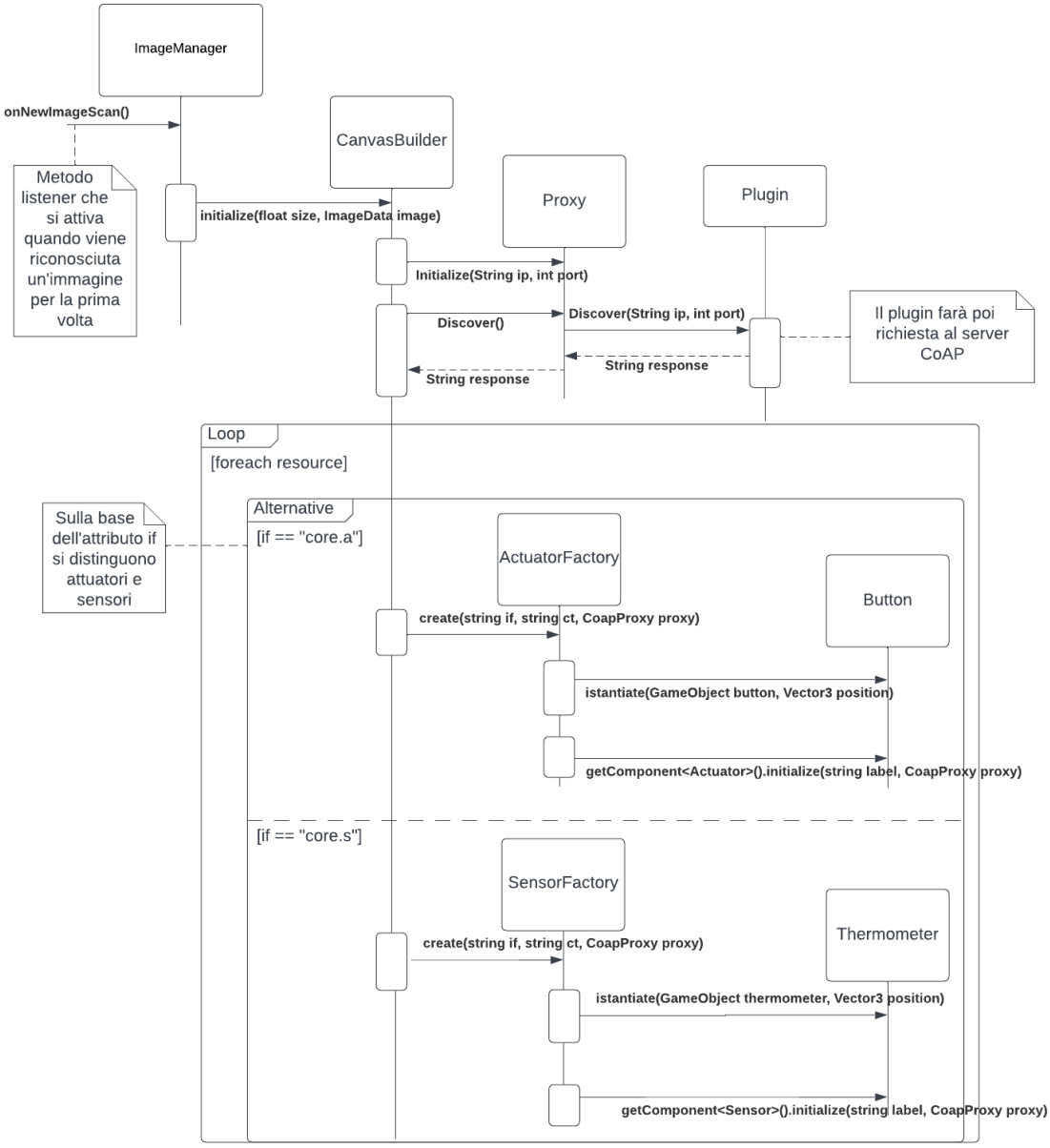

Figura 3.7. *Sequence Diagram per il processo di discovery e istanziazione delle interfacce*

Come evidenziato dall'activity diagram in Figura 3.5, CanvasBuilder delega l'istanziazione delle interfacce ai factory Actuator e Sensor (a seconda del valore dell'attributo if), i quali hanno la reference diretta agli oggetti specifici da istanziare. Queste poi istanzieranno la corrispondende componente grafica a seconda del valore rt (in riferimento al codice 3.2).

Infine si fa uso dell'attributo title per impostare il label di ogni interfaccia. Deve

essere un nome breve ma che faccia intendere subito all'utente cosa mostri o cosa faccia.

Listing 3.3. Metodo di istanziazione interfacce di CanvasBuilder

```
1  public void instantiateInterfaces()
2  {
3    factory.initialize(discoveryDict, coapProxy);
4    monitor.GetComponent<MonitorDecorator>().initialize(coapProxy);
5
6    foreach (KeyValuePair<string, string> entry in discoveryDict)
7    {
8      string ifType, label, rt, uri;
9      rt= entry.Value.Split(',')[0];
10     label = entry.Value.Split(',')[1];
11     ifType = entry.Value.Split(',')[2];
12     uri = entry.Key.Substring(1);   //tolgo la "/" all'inizio del
           nome della risorsa
13     if (ifType == "core.a")
14       factory.instantiateActuator(rt, uri, label);
15
16     if (ifType == "core.s")
17     {
18       factory.instantiateSensor(rt, uri, label);
19       monitor.GetComponent<MonitorDecorator>().setUri(uri);
20       monitor.GetComponent<MonitorDecorator>().setLabel(label);
21     }
22   }
23 }
```

3.5 Plugin

Come anticipato, dovendo ricorrere a Java per comunicare con il server, è stato adottato il sistema native plugin di Unity, inserendo una libreria precompilata nel progetto.

Il codice è molto semplice, in quanto composto solo di una classe, all'interno della quale sono presenti solo metodi static; questo perché seppure è possibile istanziare classi native non statiche e chiamarne i metodi da Unity, il processo sarebbe stato molto più complicato da integrare.

Bisogna passare ogni volta i dati necessari per la comunicazione al server, così facendo inoltre non è necessario istanziare più volte la classe per ogni canvas.

Siccome è parte del remote proxy, i metodi sono definiti allo stesso modo che alla

classe CoapProxy in C#, ma mentre quest'ultimo si occupa di chiamare il plugin, il primo chiama direttamente il server, ma è un semplice client che svolge le operazioni minime per contattarlo, con tutti i vari controlli.
Nel listing 3.4 è riportato uno dei metodi della classe del plugin (GET), per dare un'idea del funzionamento, mentre nel 3.5 viene mostrato in modo semplificato come avviene una chiamata ai metodi del plugin (si noti che solo la classe CoapProxy può effettuare questo tipo di operazione).

Listing 3.4. metodo GET del plugin in Java

```
1  public static String get(String ip, String resource) {
2          String response= new String();
3          CoapClient client;
4          try{
5              client=getClient(ip, resource);
6          }catch (URISyntaxException e) {
7              Log.d("[DEB] error","Invalid URI: " + e.getMessage());
8              response= "invalid URI" + e.getMessage();
9              return response;
10         }
11         Request request = new Request(CoAP.Code.GET);
12         request.setOptions(new OptionSet().setAccept(
               MediaTypeRegistry.APPLICATION_SENML_JSON));
13         request.setConfirmable(true);
14
15         try {
16             Log.d("[DEB]", "Calling server");
17             CoapResponse coapResponse = client.advanced(request);
18             Log.d("[DEB]", "got a response");
19
20             if (coapResponse != null) {
21                 response = coapResponse.getResponseText();
22             } else {
23                 response = "No response received.";
24             }
25
26         } catch (ConnectorException | IOException e) {
27             response = "Got an error: " + e;
28             return response;
29         }
30
31         Log.d("[DEB]", response);
32         client.shutdown();
33         return response;
34     }
```

Listing 3.5. Chiamata al metodo GET del plugin (da CoapProxy)

```
1  public class CoapProxy : MonoBehaviour, CoapManager
2  {
3      const string pluginName = "com.example.coapplugin.
           PostGetClient";
4      static AndroidJavaClass _pluginClass;
5      //... altro codice
6
7      public static AndroidJavaClass PluginClass
8      {
9          get
10         {
11             if (_pluginClass == null)
12             {
13                 _pluginClass = new AndroidJavaClass(pluginName);
14             }
15             return _pluginClass;
16         }
17     }
18
19     public string get(string resource)
20   {
21         string response = PluginClass.CallStatic<string>("get",ip,
               resource);
22         return response;
23     }
24 }
```

Capitolo 4

Use cases

Mentre nel precedente capitolo è stato mostrato l'uso generale del software, in questo si presenterà qualche server di esempio per mostrare in pratica il funzionamento dell'applicazione.

4.1 Server

Se il client è solo un'interfaccia attraverso la quale possiamo interagire con il digital twin, il server CoAP è ciò che rappresenta il dispositivo stesso.
Per l'occasione sono stati creati diversi server ad hoc, tutti nel campo della domotica per essere facilmente compresi:

- **Coffee machine:** una macchinetta del caffè, in grado di mostrare il tipo di capsule presenti (per ognuna il suo numero), aggiungere capsule di tipo short, impostare la quantità di zucchero richiesta, controllare la temperatura interna e la quantità di acqua presente.
- **Air conditioner:** condizionatore che mostra temperatura esterna ed umidità, permette di cambiare modalità (da una lista di 4), e di impostare la temperatura.
- **Led light:** luce LED smart, in grado di cambiare colore (da una lista di alcuni colori preimpostati) e la luminosità.

4.2 Client

Per il modo in cui è stata progettata l'app, seguendo i principi di generalità, non è necessario dover estendere il client se si presenta un nuovo server. Però, in caso il server sfrutti dei valori rt ignoti al client, verrebbero mostrati di default come testo 2D dal client, il che non sfrutterebbe a pieno la natura AR dell'applicazione (limitando la sua utilità effettiva). Nel caso peggiore, se un attuatore che ha bisogno di valori di input per funzionare (ad esempio un valore numerico) non viene riconosciuto dal client, di default verrà istanziato un bottone, il quale può soltanto inviare richieste POST vuote, difatti non riuscendo ad interagire correttamente con il server.
Al momento sono state implementate le seguenti interfacce specifiche:

- Termometro per indicare la temperatura (con qualsiasi unità di misura).
- Capsule di dimensioni differenti per indicare il tipo di caffè (lungo, corto, normale).
- Slider per input numerici in un certo range.
- Barra di caricamento per indicare valori percentuali.
- Switch on/off per attivare/disattivare un dispositivo o una sua funzione, o in generale per valori booleani.

Figura 4.1. *Coffee machine screenshot*

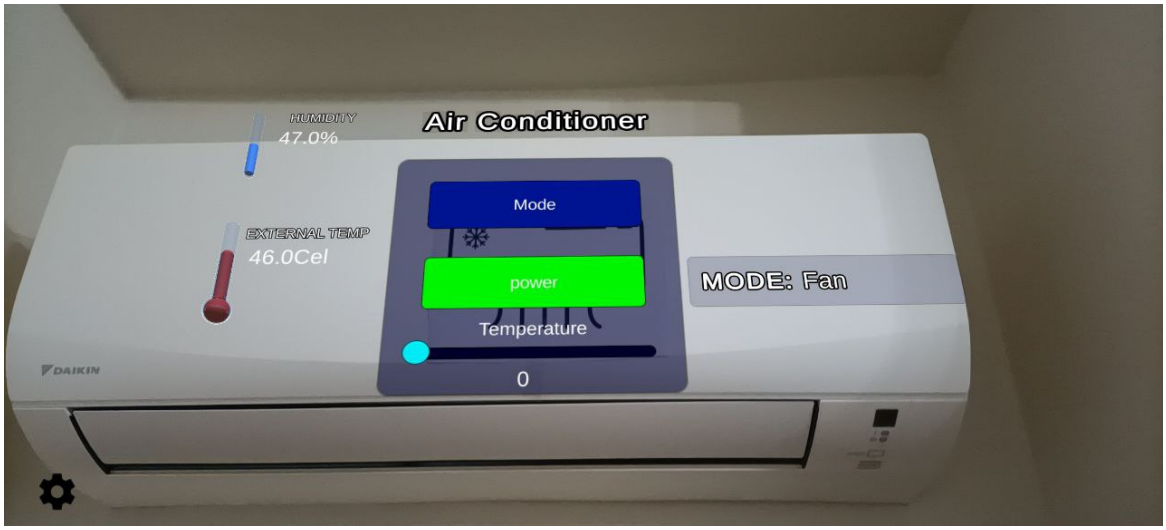

Figura 4.2. *Air conditioner screenshot*

4.3 Risorse sconosciute

Uno degli obiettivi del modulo del client è quello di funzionare con qualsiasi server ed interfacciarsi a quante più risorse possibili. Nel caso in cui un attributo rt non sia conosciuto, semplicemente l'app mostrerà un monitor per i sensori, ed un pulsante default per gli attuatori (non è detto però che il pulsante riesca ad interagire correttamente con la risorsa, per i motivi precedentemente discussi). In questo modo possiamo comunque ottenere informazioni nel caso in cui il digital twin abbia solo risorse sconosciute, oppure nel caso in cui se ne conoscono alcune, ma non tutte.

Di seguito è riportato un esempio di un server del quale non è possibile riconoscere alcuna risorsa, ma con il quale è comunque possibile interagire correttamente:

Il server in questione è associato ad una lampadina LED, il quale digital twin permette di cambiare il colore, la luminosità, e di accendere/spegnere la lampadina. Con soli monitor e pulsanti si è perfettamente in grado di interagire con il gemello reale e di visualizzare tutti i suoi dati, anche se non si sfruttano al meglio le potenzialità della realtà aumentata.

L'attuatore per la luminosità ad esempio poteva essere controllato tramite uno slider, mentre in questo caso premendo il pulsante brightness, si scorre una lista di 4 valori

percentuali (25%, 50%, 75% e 100%). Se il server avesse accettato solo numeri da 0 a 100 non sarebbe stato possibile interagire tramite richieste post vuote, e quindi il pulsante default sarebbe stato inutilizzabile.

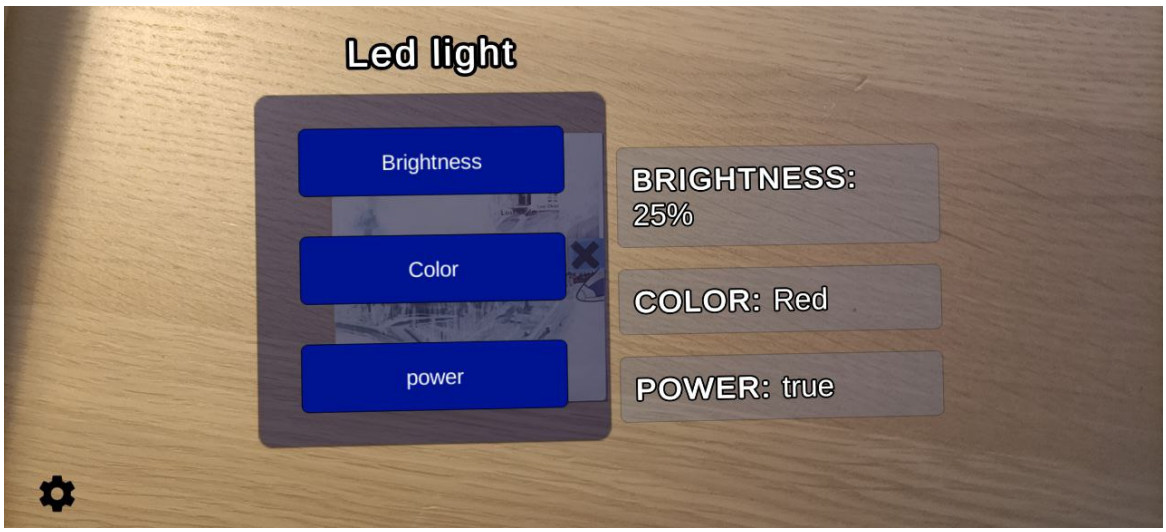

Figura 4.3. *LED server con attributi RT sconosciuti*

Capitolo 5

Conclusioni

In questo capitolo conclusivo verranno esposte possibili applicazioni reali in diversi ambiti per il software realizzato, e si discuterà della sua effettiva utilità nella vita di tutti i giorni, valutando vantaggi e svantaggi. Si ipotizzeranno infine possibili scenari futuri in cui IoT, digital twin e AR potrebbero avere successo e diffondersi su larga scala.

5.1 Applicazioni

La generalità e la natura dinamica dell'applicazione prodotta la rende adatta a diversi campi di utilizzo, aiuta anche la facile estensibilità delle interfacce speciali messe a disposizione.
Per questo motivo, sono stati immaginati possibili scenari d'uso concreti di questo tipo di interfaccia utente, per i seguenti campi:

5.1.1 Ambito industriale

In ambito industriale, come anticipato nel capitolo 1, si ha spesso a che fare con macchinari complessi, magari con un elevato numero di sensori particolari.
Visualizzare i dati in modo intuitivo fornisce un grosso aiuto per la loro interpretazione, in certi casi aiutando personale nuovo a comprenderli facilmente.
Un digital twin di tali sistemi industriali, che fornisce anche supporto decisionale, potrebbe anche mostrare le proprie considerazioni direttamente sull'interfaccia AR.

La possibilità di aggiungere nuove interfacce specifiche diventa un aiuto essenziale in questo caso.

5.1.2 Domotica

L'esempio della macchinetta del caffè è un caso perfetto di domotica, essendo un dispositivo presente in molte case.
Questo concetto è facilmente estendibile alle luci smart, ai condizionatori, sistemi multimediali, e tutti i possibili elettrodomestici coi quali si può interagire in modo analogo.
Seppur interagire con questi dispositivi in modo diverso non porta benefici evidenti come nel caso industriale, può comunque essere un caso d'uso interessante, o una spinta verso l'adozione di sistemi IoT per automatizzare e gestire in modo più efficiente ed automatico i dispositivi casalinghi.

5.1.3 Smart cities

Per non limitarsi all'uso personale, si può estendere l'applicazione precedente al caso delle smart cities e delle infrastrutture pubbliche: si immagini ad esempio di inquadrare una fermata dell'autobus e avere la possibilità di visualizzare i prossimi arrivi o comprare il biglietto, oppure inquadrare un cartello stradale per ottenere indicazioni su applicazioni di mappe, ed eventuale situazione sul traffico in tempo reale.
Questi sono solo alcune ipotesi di applicazioni utili che potrebbero essere applicate sulla base di dati pubblici che molte città offrono già adesso.

5.2 Limiti attuali e tecnologie future

Usare un'interfaccia AR potrebbe risultare inizialmente interessante e vantaggiosa per i motivi sopra elencati, ma nel lungo termine (o anche dopo un paio di usi), il tempo richiesto per inquadrare un immagine, e il dover tener fermo il dispositivo per inquadrarla correttamente, fanno diventare l'esperienza tediosa e scomoda, rendendo più vantaggioso l'uso di una normale applicazione classica in 2D.
Di fatti smartphone e tablet, nonostante siano un buon modo per provare la realtà aumentata siccome sono dotati di tutti i sensori richiesti, e di sufficiente capacità computazionale, non sono i dispositivi giusti per rendere che queste tecniche diffuse ed utilizzate nel quotidiano.
Sono invece più indicati i *Mixed Reality Glasses* già introdotti nei capitoli precedenti; questi, mostrando direttamente gli elementi virtuali senza dover fisicamente tenere in mano uno smartphone, ma semplicemente *osservando* l'immagine da riconoscere. L'esperienza d'uso potrebbe compararsi a quella degli ologrammi, diventando nettamente più naturale, superando in comodità l'uso di interfacce 2D classiche.
Possono anche essere dotati di ulteriori sensori speciali come ad esempio quelli per il tracking degli occhi, o per il tracking delle mani (per poter interagire direttamente con i gesti della mano, riconosciuti da sensori o dalla telecamera con algoritmi di image recognition).
Allo stato dell'arte attuale, questi dispositivi sono ancora troppo costosi ed ingombranti da poter essere adottati su larga scala anche da utenti comuni, e nel frattempo vengono utilizzati in rari casi in ambiti industriali.
Un altro importante problema è dato dall'elevata eterogeneità dei dispositivi IoT, spesso dotate di soluzioni proprietarie abbastanza chiuse (per incentivare l'idea di "ecosistema" di prodotti dell'azienda produttrice, sfavorendo l'interoperabilità tra dispositivi IoT diversi).

5.3 Sviluppi futuri

Di seguito verranno esposte possibili nuove vie di sviluppo per migliorare il sistema, ed integrare nuove funzionalità che ne estendano la compatibilità con altri tipi di server:

5.3.1 Web of Things

Oltre al concetto di Internet of Things, esiste quello di *Web of Things*, definito dal World Wide Web Consortium.[1] Esso si pone lo scopo di garantire l'interoperabilità tra dispositivi IoT interconnessi (spesso molto eterogenei in tutti gli aspetti), definendo quattro "*building blocks*" che devono essere implementati:

- **Thing Description:** una descrizione di un oggetto IoT, in modo da fornire metadata fruibile sia da umani che da altri dispositivi (M2M). è come se fosse la pagina index.html di un sito Web, ossia il punto di ingresso fondamentale del dispositivo.
- **Binding Templates:** specifica per ogni protocollo come deve avvenire l'interazione WoT.
- **Scripting API:** bisogna mettere a disposizione un'API che segue le specifiche della thing description.
- **Sicurezza e Privacy.**

Seguendo questi standard, si potrebbe ovviare al problema di dover per forza essere a conoscenza del valore rt per istanziare interfacce particolari, in quanto potrebbero essere standardizzate indipendentemente dall'azienda produttrice di quel dispositivo IoT.

5.3.2 Digital Twins e AR Glasses

L'argomento digital twin è stato trattato maggiormente in via teorica, ma sono presenti librerie e framework che si occupano di gestire tutto ciò che riguarda questo argomento (come ad esempio Eclipse Ditto di Java), permettendo di seguire tutto il ciclo di vita di un prodotto fisico, utile soprattutto per l'ambito industriale.

Come discusso, un'implementazione del progetto su AR Glasses (come Hololens di Microsoft), non sarebbe troppo complicata grazie al framework AR Foundation di Unity, e migliorerebbe nettamente l'esperienza d'uso.

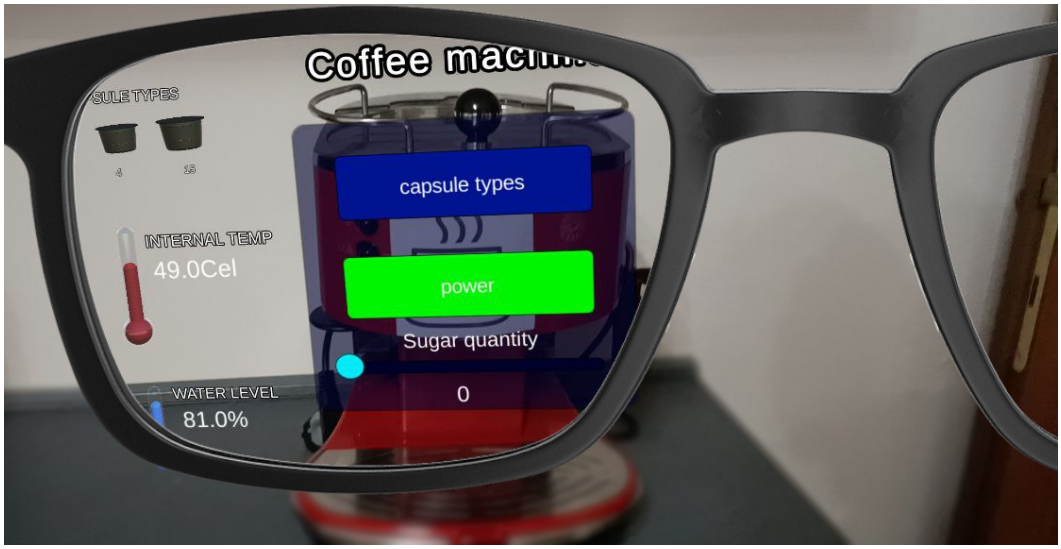

Figura 5.1. *Concept di occhiali AR con il software progettato.*

5.3.3 Aggiunta di altre interfacce specifiche

Aggiungere nuovi sensori 3d e attuatori, che possano coprire più server possibili ed ottenere un'interazione migliore in realtà aumentata.
Essendo questo un punto molto importante, è probabile che questo tipo di software potrebbe trarre molto vantaggio da una licenza di distribuzione di tipo open source, in quanto chiunque potrebbe modificare ed estendere il software, aggiungendo interfacce aggiuntive (seguendo la struttura già impostata), oppure aggiungere nuove funzionalità di altro tipo (come compatibilità con altri dispositivi client o altri server), che gioverebbero soltanto alla generalità del software e all'esperienza d'uso per gli utenti.

Bibliografia

[1] Consortium, W. W. W. *Web of Things documentation*. Available from: https://www.w3.org/WoT/documentation/.

[2] Durrant-Whyte, H. and Bailey, T. Simultaneous localization and mapping: part i. *IEEE Robotics & Automation Magazine*, **13** (2006), 99. doi:10.1109/MRA.2006.1638022.

[3] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley (1994).

[4] Google. *Overview of ARCore and supported development environments* (2022). Available from: https://developers.google.com/ar/develop.

[5] IBM. What is a digital twin? Available from: https://www.ibm.com/topics/what-is-a-digital-twin.

[6] Jennings, C., Cisco, Shelby, Z., ARM, Arkko, J., Keranen, A., Ericsson, Bormann, C., and TZI, U. B. Sensor measurement lists (senml). RFC 8428, RFC Editor (2018). Available from: https://datatracker.ietf.org/doc/html/rfc8428.

[7] Shelby, Z., ARM, Hartke, K., Bormann, C., and TZI, U. B. The constrained application protocol (coap). RFC 7252, RFC Editor (2014). Available from: https://datatracker.ietf.org/doc/html/rfc7252.

[8] Shelby, Z. and Sensinode. Constrained restful environments (core) link format. RFC 6690, RFC Editor (2012). Available from: https://datatracker.ietf.org/doc/html/rfc6690.

[9] Unity. *Documentazione su native plugins*. Available from: https://docs.unity3d.com/Manual/PluginsForAndroid.html.

[10] Zhu, Z., Liu, C., and Xu, X. Visualisation of the digital twin data in manufacturing by using augmented reality. *Procedia CIRP*, **81** (2019), 898. 52nd CIRP Conference on Manufacturing Systems (CMS), Ljubljana, Slovenia, June 12-14, 2019. doi:https://doi.org/10.1016/j.procir.2019.03.223.