ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

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

ANALISI DELLE PERFORMANCE DI SISTEMI DI MIXED REALITY COOPERATIVI IN RETE LOCALE

Elaborato in
SISTEMI EMBEDDED E INTERNET OF THINGS

Relatore
Prof. ALESSANDRO RICCI

Presentata da
FILIPPO VENTURINI

Corelatore
Dott. SAMUELE BURATTINI

Anno Accademico 2021 – 2022

Ai miei genitori,
per tutte le volte che ci siete stati per me
e che io ci sarò per voi.

Indice

Introduzione

Lo sviluppo tecnologico sta portando il mondo che conosciamo a subire delle evoluzioni sotto diversi aspetti, in molti ambiti ci si sta avvicinando a tecnologie che non molto tempo fa venivano considerate di stampo fantascientifico, mentre oggi sono tutt'altro che lontane e immaginarie e presto impatteranno la nostra vita di tutti i giorni.
Siamo sempre stati abituati fin dal principio ad interagire con l'ambiente che ci circonda, utilizzando gli oggetti fisici presenti attorno a noi per soddisfare le nostre esigenze, ma se esistesse di più di questo?
Se fosse possibile avere attorno a noi oggetti che non sono propriamente corpi fisici, ma che hanno un comportamento coerente con l'ambiente circostante e non venisse percepita la differenza tra essi e un vero e proprio oggetto?
Ci si sta riferendo a quella che oggi viene chiamata Mixed Reality, una realtà mista resa visibile tramite appositi dispositivi, in cui è possibile interagire contemporaneamente con oggetti fisici e oggetti digitali che vengono chiamati ologrammi.
In questi sistemi la parte digitale aggiunge contenuto informativo all'ambiente fisico, fornendo varie funzionalità che possono avere un ruolo fondamentale in vari ambiti: quello della comunicazione, quello lavorativo, videoludico e tanti altri. Possiamo immaginare un futuro non così lontano, in cui gli ologrammi avranno una profonda pervasività nella nostra vita quotidiana, verranno sicuramente utilizzati per la divulgazione di informazioni, modificheranno la collaborazione a livello lavorativo con colleghi anche molto distanti da noi e in generale avranno un impatto determinante percepibile da chiunque.
Con un ottica proiettata verso il futuro si evince che questo tipo di sistemi non possono semplicemente essere applicativi locali ai dispositivi, ma devono possedere un'architettura più elaborata, che consenta la realizzazione di un vero e proprio mondo digitale che esiste indipendentemente dal fatto che qualcuno lo stia visualizzando, ed evolve il proprio stato permettendo alle entità presenti al suo interno di sviluppare particolari comportamenti.
Facendo riferimento a questo concetto, un aspetto fondamentale che riguarda questa tipologia di sistemi distribuiti è sicuramente la collaborazione.

Si può immaginare un sistema che mantiene questo mondo digitale e ne permette la visualizzazione a più utenti contemporaneamente, permettendo un'interazione di tipo collaborativo all'interno di esso.
Nel primo capitolo della tesi vengono quindi fissati i concetti necessari per poter affrontare queste tematiche, facendo chiarezza tramite documentazione scientifica sulla corretta interpretazione delle varie nozioni. Successivamente verranno analizzate le tecnologie presenti al giorno d'oggi sia lato hardware che lato software per poter realizzare delle applicazioni nell'ambito della Mixed Reality.
Nel secondo capitolo ci si concentrerà invece sull'argomento principe trattato in questa tesi, ovvero la collaborazione, esaminando lo stato dell'arte e analizzando la struttura architetturale delle tecnologie presenti al giorno d'oggi che permettono la condivisione di esperienze di realtà mista.
Passando alla parte progettuale, l'obbiettivo di questa tesi è riuscire a progettare e sviluppare un sistema che permetta la realizzazione di un mondo digitale permanente, che è possibile visualizzare in condivisione e verificarne le prestazioni.
Nel terzo capitolo viene quindi affrontata una prima parte di progettazione infrastrutturale, sia a livello di pattern di progettazione che a livello di rete, collocando il progetto in un ambito applicativo d'esempio.
Nel quarto capitolo viene dunque presentato l'effettivo sviluppo dell'applicativo, realizzando un sistema che a differenza delle tecnologie allo stato dell'arte è totalmente indipendente da servizi cloud, e utilizza TCP come protocollo di comunicazione.
Nel quinto e ultimo capitolo, si riscontrano i risultati di quanto prodotto nelle fasi precedenti, ottenendo delle misurazioni che riportano dati oggettivi sulle prestazioni del sistema. Vengono testate le performance dell'applicativo in vari scenari significativi di complessità crescente e vengono fatte valutazioni sui risultati ottenuti, traendone delle conclusioni.

Capitolo 1

Concetti introduttivi

Nel primo capitolo verranno illustrati vari concetti di base, necessari per poter procedere ad esaminare il caso di studio preso in considerazione.

Inizialmente è necessario esplorare il concetto di Extended Reality per esaminare sia intuitivamente che formalmente che cosa si intende per realtà estesa. Successivamente occorre introdurre e differenziare i concetti di Virtual, Mixed e Augmented Reality che spesso vengono erroneamente utilizzati come sinonimi, ma presentano differenze sostanziali sia a livello tecnologico che concettuale. Una volta definite queste nozioni chiave si entrerà nel merito delle tecnologie utilizzate in questo caso di studio, sia a livello hardware che a livello software.

1.1 Extended Reality

La nozione di **Extended Reality** (XR) può essere sintetizzata intuitivamente con il concetto di realtà estesa, infatti essa può essere immaginata a tutti gli effetti come un mondo fisico in cui un utente si trova, con integrata un'estensione virtuale che aggiunge contenuto informativo.
L'estensione virtuale del mondo fisico può avere varie profondità e sfumature ed è per questo che il termine Extended Reality viene considerato molto generico, è necessario quindi esaminare la relazione che intercorre in un sistema XR tra il mondo fisico e il mondo virtuale per poter definire i vari livelli di estensione della realtà.
Per effettuare quest'analisi è utile fare riferimento a quello che viene chiamato *"Reality-Virtuality Continuum"* concepito da Milgram e Kishino nel 1994 (fig. 1.1).

1.1.1 Il Reality-Virtuality Continuum

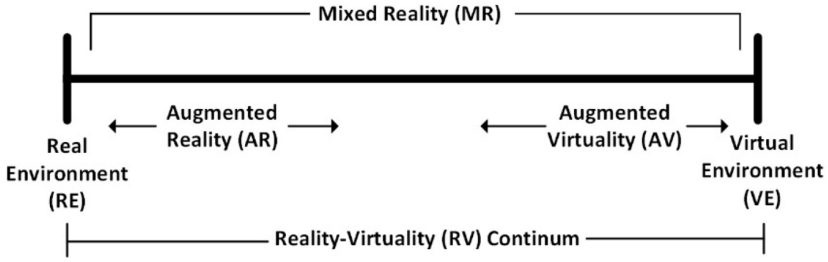

Figura 1.1: Reality-Virtuality Continuum.

Il **Reality-Virtuality Continuum** mostra la relazione tra il mondo fisico e quello virtuale in un ambiente di Extended Reality. Al primo estremo del RV Continuum notiamo il mondo fisico, ovvero l'ambiente in cui è situato l'utente mentre all'estremo opposto è presente l'ambiente virtuale, cioè un vero e proprio mondo digitale nel quale l'utente è completamente immerso. La nozione di mondo virtuale corrisponde al concetto di **Virtual Reality** (VR) che verrà approfondito in seguito.
Proseguendo da sinistra, molto vicino all'ambiente fisico, notiamo quella che viene chiamata **Augmented Reality** (AR) che rappresenta l'estensione della realtà meno pervasiva e immersiva. Questa tipologia di estensione della realtà può essere vista come una struttura a due livelli, il livello base consiste nel mondo fisico, a cui viene sovrapposto il livello virtuale tramite l'aggiunta di informazioni digitali (tipicamente immagini o ologrammi).
Viceversa, procedendo verso l'estremo opposto, viene definito un'altro concetto ovvero **l'Augmented Virtuality**(AV) che presenta sempre una struttura a due livelli, in cui però è la parte virtuale a fare da base e sono oggetti appartenenti al mondo fisico che vengono inseriti al suo interno [1].
Infine viene mostrata la **Mixed Reality** (MR) che concettualmente presenta caratteristiche miste e consiste in una realtà ibrida, nella quale è possibile interagire sia con il mondo fisico che con quello virtuale. Quest'ultima definizione incapsula l'ambito applicativo in cui è stato attuato il caso di studio che verrà successivamente descritto.
Riassumendo quindi, l'Extended Reality è un concetto generico che si riferisce all'estensione del mondo fisico, ha diversi livelli di pervasività del layer virtuale all'interno di quello fisico e vengono distinti utilizzando i concetti di AR, MR e VR che sono sottoinsiemi dell'XR (fig. 1.2).
Si procede quindi ad illustrare singolarmente questi sottoinsiemi e ad evidenziarne le differenze sia a livello concettuale che tecnologico.

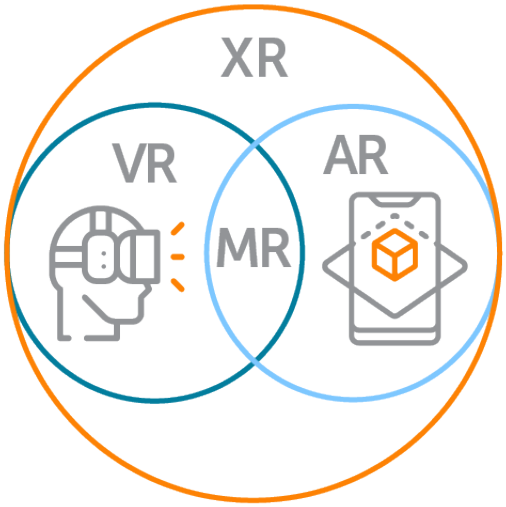

Figura 1.2: Extended, Virtual, Mixed e Augmented Reality.

1.1.2 Virtual Reality

La **Virtual Reality** rappresenta l'immersione più profonda nella parte digitale della realtà estesa.
Si parla infatti di tecnologie che permettono di far entrare l'utente in un vero e proprio mondo digitale in tre dimensioni, all'interno del quale vengono ricevuti stimoli ed è possibile operare non avendo effetti nel mondo fisico che ci circonda.
Lo sviluppo tecnologico permette a questo tipo di ecosistemi di incrementare costantemente l'immersività dell'utente. Inizialmente l'esperienza VR era concentrata sulla parte visiva, tramite l'utilizzo di visori (es: Oculus Rift) che una volta indossati permettono la visualizzazione del mondo virtuale, successivamente l'avanzamento tecnologico ha portato un ampliamento dell'immersività con la produzione di controller per l'interazione tramite le mani, guanti per rilevare con precisione le gesture, sistemi di navigazione tramite piattaforme che permettono il movimento all'interno del mondo virtuale, fino ad arrivare a veri e propri sistemi di rilevamento degli arti [2].
L'esperienza VR presenta quindi una struttura hardware di supporto consistente, che rende l'immersività del sistema destinata ad aumentare.

1.1.3 Augmented Reality

Con l'**Augmented Reality** ci si orienta invece su una minore pervasività della parte digitale nel mondo fisico.
Riferendosi al nome, i sistemi di AR hanno come obbiettivo quello di estendere la realtà aumentandola, aggiungendo del contenuto informativo virtuale.
Si parla infatti tipicamente dell'inserimento nel mondo fisico di ologrammi o immagini, con cui non è possibile interagire ma che hanno lo scopo di aggiungere informazioni coerenti con il contesto applicativo.
L'utilizzo più comune di questi sistemi riguarda l'aumento della realtà nella dimensione visiva, ma questo concetto può potenzialmente essere espanso anche agli altri sensi. L'AR potrebbe aumentare l'olfatto, il tatto o anche l'udito, sostituendo il senso preso in considerazione, ad esempio utilizzando segnali audio per gli utenti non vedenti, oppure informazioni visive per gli utendi non udenti, migliorando così anche l'accessibilità del sistema [3].
A livello hardware è presente una profonda differenza tecnologica rispetto alla Virtual Reality, nell'AR vengono infatti solitamente utilizzate le fotocamere di dispositivi mobili come smartphone o tablet che sono tipicamente molto più economici dell'hardware necessario per un'esperienza VR. Una soluzione meno economica per l'Augmented Reality sono gli smart glasses (es: Amazon Echo-Frames) che rendono l'esperienza sicuramente più immersiva.

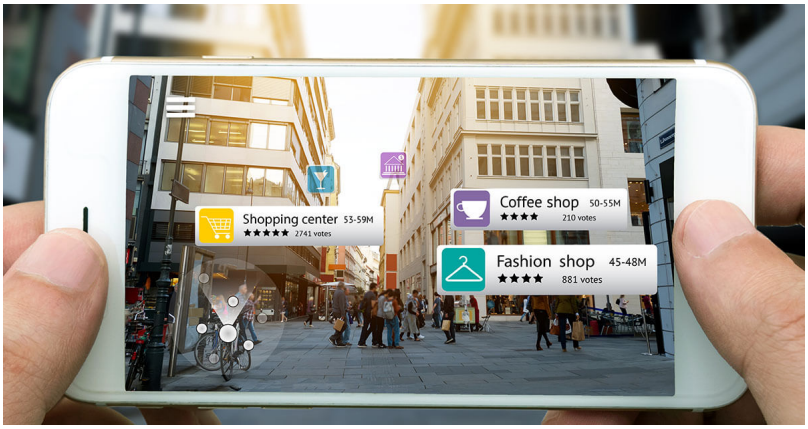

Figura 1.3: Esempio di Augmented Reality.

1.1.4 Mixed Reality

Infine è necessario introdurre l'ultimo concetto fondamentale che è alla base del caso di studio che verrà trattato di seguito, ovvero la **Mixed Reality**. Quest'ultima categoria di estensione della realtà può essere vista come un'intersezione tra le due precedentemente illustrate (fig. 1.2).
Si considera infatti una vera e propria realtà mista: l'utente si trova in un ambiente fisico, in cui vengono inseriti elementi digitali (tipicamente ologrammi 3D) con cui è possibile interagire.
Viene quindi realizzato un ecosistema nel quale è possibile operare indistintamente con oggetti appartenenti al mondo fisico e a quello virtuale ed è inoltre presente un'interazione bidirezionale: non solo il mondo virtuale ha effetti sul mondo fisico visualizzando per esempio degli ologrammi, ma la parte virtuale può essere influenzata e variare il proprio comportamento in base agli eventi che si verificano nel mondo fisico.
La MR presenta quindi un ambiente in cui coesistono un layer fisico e un layer virtuale, l'obbiettivo è riuscire a creare un collegamento tra di essi che sia quasi impercettibile: un utente dovrebbe avere la possibilità di interagire sia con oggetti fisici che virtuali senza distinzione.
Questa tipologia di realtà estesa permette di accennare il concetto di Mirror World, ovvero un vero e proprio mondo che associa ad ogni entità fisica o ologramma appartenente al sistema di MR, una controparte digitale che la rappresenta ad un livello di astrazione più alto e ha una comunicazione bidirezionale con il mondo fisico [4].
Tipicamente i sistemi di MR sono resi possibili tramite l'utilizzo di visori con lenti trasparenti che proiettano gli ologrammi 3D nel mondo circostante (es: Microsoft HoloLens).

Figura 1.4: Esempio di Mixed Reality.

1.2 Mixed Reality: zoom concettuale

Nelle sezioni precedenti si è fatta una distinzione tra XR, AR, VR e MR, esaminando differenze e analogie e fornendo una visione d'insieme sul panorama dell'Extended Reality.
Ora è necessario esaminare nel dettaglio la Mixed Reality, essendo essa il contesto in cui è stato svolto questo caso di studio. Verranno introdotte nozioni specifiche sulla realtà mista ed esaminati i meccanismi che formano la base di questo tipo di tecnologie.

1.2.1 Ologrammi

Come introdotto precedentemente la MR consiste in una coesistenza nello stesso ambiente, di un layer fisico e un layer virtuale, con cui l'utente può interagire indistintamente. Il layer virtuale presenta una tecnologia fondante, ovvero l'ologramma.
Il termine 'Hologram' è stato coniato da Dennis Gabor che unì le parole greche "holos" che significa "tutto" e "gram" che significa "messaggio" [5].
Quindi per introdurne il concetto, un ologramma consiste in un'immagine in tre dimensioni, visibile ad occhio nudo, che viene proiettata nel mondo fisico tramite appositi dispositivi. Un ologramma ha tipicamente lo scopo di aggiungere contenuto informativo di tipo visivo in un ambiente e può essere o meno manipolabile dall'utente, in base alle esigenze e alle tecnologie che si stanno utilizzando. I meccanisimi di realizzazione degli ologrammi vengono definiti come metodi di *"three-dimensional imaging"* e permettono tramite la diffrazione della luce di proiettarli nel mondo fisico.
In generale un ologramma può essere un oggetto di qualsiasi tipo, a partire da semplici forme geometriche come cubi o sfere, fino a rappresentare oggetti complessi come prototipi di automobili o avatar associati a persone fisiche.
Il concetto di ologramma è differente rispetto al contenuto digitale che viene aggiunto al mondo fisico in sistemi di Augmented Reality. Quando si parla di AR, come già introdotto in precedenza, si necessita di dispositivi di visualizzazione della realtà aumentata come smartphone o tablet, che tramite le loro fotocamere permettono la renderizzazione di immagini digitali sovrapposte all'ambiente circostante.
Un ologramma in un'esperienza di Mixed Reality però è molto di più di questo, esso ha l'ambizione di risultare a tutti gli effetti come un oggetto appartenente al mondo fisico, visibile ad occhio nudo oppure tramite appositi visori ed eventualmente permette all'utente di interagire con esso manipolandolo.

1.2.2 Ambiente e Spatial Awareness

Un altro importante concetto senza il quale non sarebbe possibile realizzare un sistema di Mixed Reality è l'ambiente. Per poter estendere il mondo fisico con un layer virtuale è necessario avere a disposizione dei meccanismi che permettano alla parte virtuale del sistema di percepire l'ambiente in cui è in esecuzione.
Questa caratteristica viene chiamata *"spatial awareness"*, ovvero **consapevolezza spaziale**. In generale nei sistemi di MR per poter tracciare l'ambiente circostante viene costruita una *"Mesh"*, cioè una vera e propria rete che rappresenta computazionalmente gli oggetti fisici che vengono rilevati dai sensori presenti nel dispositivo, come muri, tavoli pavimento o soffitto [6]. La mesh fornisce perciò una **mappa ambientale** che la parte digitale del sistema utilizza per il posizionamento o più in generale per l'interazione e la coesistenza degli ologrammi nel mondo fisico.
In generale la creazione della mappa spaziale è indispensabile quando si considera un'esperienza di Mixed Reality per diversi motivi:

- *Persistenza*: Utilizzando una mappa spaziale è possibile inserire un ologramma in una determinata posizione e anche al verificarsi di una chiusura accidentale dell'applicazione, ottenere una persistenza di esso nella posizione in cui è stato inserito.

- *Occlusione*: Caratteristica fondamentale grazie alla quale è possibile occludere, quindi nascondere alla vista, un ologramma.
  Riferendosi alla spatial awareness, si utilizzano gli elementi presenti nell'ambiente (es: muri, tavoli ecc.) per nascondere un ologramma. Considerando ad esempio un cambio di stanza dopo la creazione di un ologramma, senza l'utilizzo della mappa spaziale sarebbe possibile vedere l' ologramma creato attraverso il muro e questo comprometterebbe l'essenza dell'esperienza di Mixed Reality, in cui gli ologrammi vengono visti come oggetti appartenenti al mondo fisico e che devono quindi comportarsi come tali.

- *Navigazione*: Viene reso possibile tramite lo sguardo, far percorrere agli ologrammi creati, le superfici mappate all'interno della mesh.

- *Posizionamento*: Consente agli utenti di interagire con l'ambiente circostante, permettendo di posizionare gli ologrammi sulle superfici mappate, ad esempio appoggiando un ologramma sulla superficie di un tavolo o appendendolo ad una parete.

- *Fisica*: Permette agli ologrammi di avere un comportamento più realistico, applicando le collisioni proprie del mondo fisico con l'ambiente circostante. Un esempio potrebbe essere un ologramma che rimbalza sul pavimento e le pareti della stanza in cui si trova.

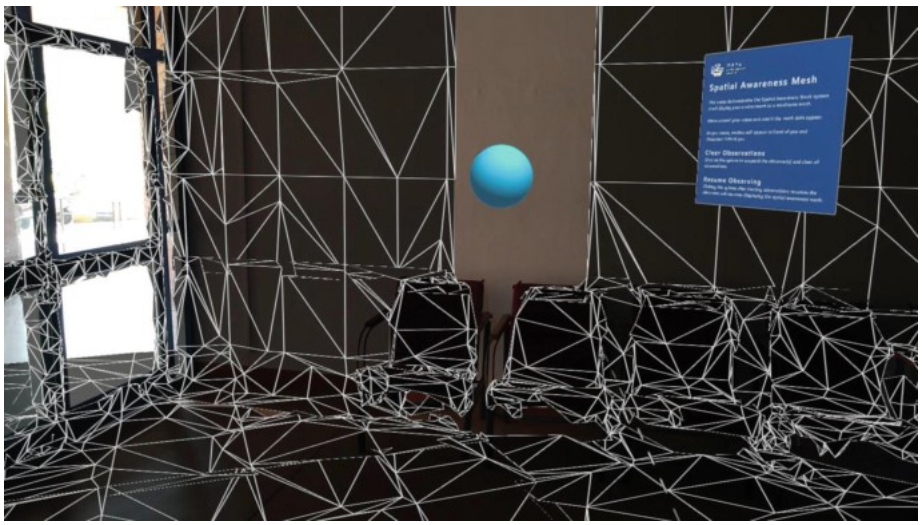

Figura 1.5: Mappatura spaziale.

1.2.3 Posizionamento degli ologrammi nell'ambiente

Una volta introdotti i due concetti chiave di un sistema di Mixed Reality, ovvero ologrammi e ambiente, è necessario esaminare come possono essere integrati, ovvero con quale logica vengono collocati gli ologrammi nell'ambiente fisico.
In generale il posizionamento degli ologrammi avviene in maniera molto semplice e intuitiva: vengono utilizzate le coordinate cartesiane con i tre assi X, Y e Z che fanno riferimento alle tre relative dimensioni spaziali.
Un dettaglio importante è l'unità di misura, i sistemi di riferimento per la Mixed Reality utilizzano i metri, ciò significa che il posizionamento di un ologramma a 2 unità dall'origine degli assi, una volta renderizzato si troverà a 2 metri di distanza.
Un'ultima precisazione importante riguarda il fatto che un sistema di riferimento tridimensionale può essere di tipo *"right handed"* oppure *"left-handed"*. In entrambi l'asse X positivo è orientato verso destra e l'asse Y positivo è orientato verso l'alto. La differenza si limita all'asse Z, che in un sistema *"right-handed"* se positivo è orientato verso l'osservatore, mentre in un sistema *"left-handed"* è orientato con verso opposto [7].

1.2.4 Interazione

Effettuando una composizione delle nozioni introdotte fino ad ora, risulta chiaro che cos'è un ologramma e in che forma si presenta, che cosa si intende per ambiente e come viene percepito e interpretato dal layer virtuale della realtà mista e infine come vengono integrati gli ologrammi all'interno di esso. Se realizzassimo un sistema di questo tipo ci troveremmo di fronte ad un ambiente di realtà mista effettivo?
Sicuramente avremmo un ecosistema in cui sono presenti sia componenti del mondo fisico che componenti virtuali, ma manca una parte fondamentale che caratterizza il concetto di Mixed Reality e lo distingue effettivamente da tutte le altre tipologie di estensione della realtà: l'interazione con gli ologrammi.
In generale un sistema di MR mette a disposizione varie tipologie di interazione, la cui complessità e precisione dipende ovviamente dalle tecnologie che vengono utilizzate.
Di seguito si illustrano brevemente le varie tipologie di interazione [8]:

- *Sguardo*: Lo sguardo è un metodo fondamentale all'interno di sistemi di MR per orientarsi e svolgere anche le operazioni più semplici. Lo sguardo viene identificato in letteratura come un metodo di puntamento, infatti viene considerato simile al mouse di un PC, essendo entrambi dispositivi con a disposizione un cursore che ne identifica lo spostamento. In questi sistemi tipicamente non viene utilizzato l'effettivo movimento degli occhi per muovere il cursore ma si considera lo spostamento della testa.

- *Gestures*: Oltre allo sguardo, ciò che permette ad un utente di sfruttare effettivamente un sistema di MR sono le gestures. Per gestures si intende l'utilizzo delle mani, che vengono rilevate dal dispositivo in uso tramite algoritmi di hand tracking, per manipolare gli oggetti virtuali presenti nel sistema. Con le gestures è possibile quindi navigare all'interno dei menu, spostare, ridimensionare o ruotare gli ologrammi e avere interazioni più dirette con l'ambiente virtuale.

- *Voce*: Un'altra tipologia di interazione con gli ologrammi in un sistema di Mixed Reality è la voce. Infatti è possibile tramite il riconoscimento vocale, pronunciare una o più parole per eseguire comandi, selezionare oggetti, oppure effettuare un rapido inserimento di testo senza utilizzare la tastiera.

- *Altro hardware*: Oltre a queste tre metodologie principali, sono stati sviluppati vari dispositivi di input basati tipicamente sull'utilizzo di Bluetooth come tastiere, mouse, gamepad, clicker e tanti altri.

1.3 Stack tecnologico d'esempio

Riassumendo il percorso di analisi del contesto applicativo, è stata fornita una visione concettuale d'insieme sul panorama dell'Extended Reality, differenziando propriamente i concetti alla base di questo ambito. Successivamente è stato fatto uno zoom dettagliato sulla nozione di Mixed Reality, analizzando con precisione i concetti e gli elementi che ne costituiscono la base, ovvero percezione dell'ambiente, ologrammi e interazioni con il layer virtuale.
Manca però un tassello importante, è necessario ridurre il livello di astrazione e concentrarsi ora sulle tecnologie che a livello pratico, abilitano lo sviluppo e l'utilizzo di questi sistemi e ne definiscono le caratteristiche. Perciò in questa sezione verrà presentato un possibile stack tecnologico che contiene tecnologie sia lato hardware che lato software e consente lo sviluppo di applicazioni di Mixed Reality.

1.3.1 Hardware: Microsoft HoloLens 2

Figura 1.6: Microsoft Hololens 2.

Nei sistemi di Mixed Reality sono presenti varie tecnologie lato hardware che permettono l'immersione in un ambiente di realtà mista. Una delle più utilizzate è sicuramente quella sviluppata da Microsoft: HoloLens 2 (fig. 1.6). HoloLens 2 appartiene alla categoria degli Head-Mounted Displays (HMD). Questi dispositivi hardware, come si intuisce dal nome, vengono indossati e hanno la funzione di mostrare all'utente l'ambiente di realtà mista in cui sta operando. Gli HMD si distinguono tipicamente in due categorie: **video see-through HMD** e **optical see-through HMD** [9].

Riferendosi alla prima categoria vengono considerati dispositivi che utilizzano una videocamera e uniscono l'ambiente percepito attraverso di essa con le immagini artificiali che corrispondono al layer virtuale della MR.
Hololens 2 appartiene però alla seconda categoria di HMD che, tramite l'utilizzo di lenti trasparenti, permette all'utente di vedere l'effettivo ambiente circostante (quindi non quello percepito dalla videocamera) e contemporaneamente anche gli ologrammi che vengono generati in esso.
Analizzando le specifiche hardware, HoloLens 2 è dotato di una Holographic Processing Unit (HPU 2.0) che abilita la visione artificiale in tempo reale a basso consumo e si occupa di eseguire gli algoritmi di visione artificiale essenziali per l'esperienza utente come: *hand tracking*, *eye gaze tracking*, *spatial mapping* e tanti altri [10].
La CPU è un Qualcomm SnapDragon 850 situato nel retro del dispositivo. Per quanto riguarda le fotocamere, che sono indispensabili e determinanti nella descrizione di queste tecnologie, il dispositivo ne possiede una di profondità e RGB e quattro a scala di grigi. A livello di memoria è dotato di 4GB di DRAM volatile e 64GB per lo storage [11].

1.3.2 Motore Grafico: Unity

Entrando ora nel dettaglio delle tecnologie lato software per lo sviluppo di applicazioni di Mixed Reality, una piattaforma fondamentale è sicuramente Unity.
Unity è un motore grafico multipiattaforma realizzato da Unity Technologies, inizialmente pensato per permettere anche ad utenti non competenti in materia di programmazione di realizzare videogiochi in 2D e 3D [14]. Con il tempo Unity è stato ampliato, permettendo di realizzare oltre a videogiochi tanti altri prodotti appartenenti a vari ambiti, tra i quali la Mixed Reality.
Un progetto Unity presenta tipicamente una o più scene che rappresentano l'ambiente in cui verrà sviluppato il progetto.
All'interno di una scena è possibile creare e manipolare uno o più oggetti 3D, è presente una camera che identifica l'utente a run-time ed è inoltre possibile modificare dinamicamente l'illuminazione.
In ambito MR, Unity viene utilizzato per realizzare la parte di View, permette quindi di creare oggetti 3D che verranno poi renderizzati come ologrammi dal dispositivo hardware su cui sarà installata l'applicazione (in questo caso HoloLens 2). La piattaforma presenta infatti un'intuitiva interfaccia che permette di creare e manipolare forme in tre dimensioni fornendo inoltre la possibilità di associare ad ogni entità creata uno o più script in C# che ne definiscono il comportamento in fase di esecuzione.

1.3.3 MRTK

Figura 1.7: Mixed Reality Toolkit.

Oltre a Unity per sviluppare applicazioni di Mixed Reality, è necessario introdurre un altro componente fondamentale: il **Mixed Reality Toolkit**.
In generale MRTK è un tool che è stato pensato per accellerare lo sviluppo cross-platform di applicazioni di MR tramite Unity [12].
MRTK è scaricabile tramite GitHub e presenta una struttura a package che incapsula le principali funzionalità messe a disposizione.
I packages disponibili sono i seguenti:

- Foundation
- Extensions
- Tools
- Test utilities
- Examples

I dettagli sulle funzionalità messe a disposizione da ciascun package sono disponibili nella documentazione Microsoft [13], ma in generale MRTK si occupa dei vari elementi fondanti di un'esperienza di Mixed Reality come il sistema di input, la Spatial Awareness (sezione 1.2.2), l'interazione con gli ologrammi e tanto altro.
Una volta scaricato il Mixed Reality Toolkit, per poterlo utilizzare è sufficiente effettuare le configurazioni richieste che sono reperibili nella documentazione di Microsoft [15].

1.3.4 OpenXR

Figura 1.8: Logo OpenXR.

"With the release of the OpenXR 1.0 specification, AR/VR developers can now create true cross-platform XR experiences."(Brent E. Insko 2019)

Citando Brent E. Insko membro di Khronos Group e responsabile della fase di progettazione e sviluppo di **OpenXR**, si procede ad introdurre l'ultimo tassello architetturale fondamentale per la realizzazione di un'applicazione di Mixed Reality. OpenXR consiste in uno standard aperto royalty-free (ovvero con limitate restrizioni sul suo utilizzo), che fornisce le API (Application Program Interface) essenziali nello sviluppo di applicazioni XR.
In generale le applicazioni in ambito Extended Reality sono destinate ad espandersi e hanno perciò l'obbiettivo di svilupparsi secondo vari aspetti come la pervasività, l'immersività e la cooperazione.
Prima dello sviluppo di OpenXR era presente un problema fondamentale (che tutt'ora non è stato completamente risolto) che ha rallentato lo sviluppo di applicazioni XR, ovvero la frammentazione. Ogni piattaforma possedeva la propria SDK (Software Development Kit) connessa al device hardware in uso. Come si può immaginare, questo ha portato varie problematiche a livello di sviluppo, per poter realizzare applicazioni cross-platform era necessario modificare il software, adattarlo al device utilizzato e in caso di errori di compilazione, essi andavano validati su tutte le piattaforme.
Grazie alle API fornite da OpenXR che incapsulano le funzionalità comuni a tutte le applicazioni XR (che sono quindi indipendenti dal device hardware), si è verificato un notevole aumento di portabilità.

Capitolo 2

Collaborative Mixed Reality

Nel primo capitolo sono state fornite le nozioni di base per orientarsi nel mondo delle applicazioni di Extended Reality, è stata presentata una panoramica sui concetti teorici fondamentali presenti in letteratura e infine sono state illustrate le tecnologie hardware e software che vengono correntemente utilizzate in questi contesti.
In questo capitolo si esaminerà nel dettaglio il fulcro teorico di questo caso di studio: le **esperienze condivise** in ambito di Mixed Reality. Verranno fornite le nozioni di base per poter comprendere questa branca riguardante la MR e verranno esaminate le tecnologie già presenti allo stato dell'arte che consentono questa tipologia di esperienze.

2.1 Esperienze condivise

2.1.1 Definizione

"A shared experience is exactly what it sounds like: seeing, hearing, or doing the same thing as someone else."

(Sean Ong e Varun Kumar Siddaraju, 2021)

Facendo riferimento alla citazione sopra indicata, è necessario introdurre il concetto di esperienza condivisa. In generale ci si riferisce al concetto di condivisione come al vivere la stessa esperienza di una o più persone, quindi condividere le percezioni sensoriali come vista e udito o più pragmaticamente interagire con gli stessi oggetti, siano essi fisici o virtuali.

2.1.2 Condivisione in sistemi di Mixed Reality

Una volta definito formalmente che cosa si intende per esperienze condivise è necessario contestualizzarlo all'interno del concetto di Mixed Reality.
Fino ad ora sono stati illustrati sistemi in cui un utente, tramite apposito hardware come ad esempio un Head Mounted Display può accedere ad un'applicazione di realtà mista, interagendo contemporaneamente con oggetti del mondo fisico e ologrammi. Ma esaminando questo esempio con una visione orientata al futuro questo non è certamente abbastanza, infatti possiamo immaginare un sistema di questo tipo in cui l'applicazione in questione è un videogioco oppure fornisce assistenza sul posto di lavoro (es: medico in sala operatoria, operaio al lavoro con macchinari ecc.). In questo modo viene reso evidente che manca una parte fondamentale: la condivisione dell'esperienza.
Per rendere più intuitivo il significato di condivisione in un ambiente di MR, possiamo immaginare più utenti dotati dell'hardware necessario, non obbligatoriamente il medesimo, che accedono allo stesso sistema di Mixed Reality e hanno la possibilità di collaborare e interagire, avendo effetti permanenti e coerenti sul sistema. In generale, considerando un applicativo di questo tipo potrebbe essere possibile inoltre avere vari permessi nell'interazione con il sistema: immaginando il caso di studio di una sala operatoria, in cui è presente un ologramma che mostra i parametri vitali del paziente, il chirurgo potrebbe modificare l'ologramma mentre il personale non abilitato avere solamente la possibilità di visualizzarlo.
Questa prospettiva introduce una vasta gamma di possibilità e applicazioni che non si limitano ad un esperienza locale effettuata da un solo utente.

Figura 2.1: Collaborative Mixed Reality.

2.1.3 Spazio e tipologia di interazione

In generale possiamo già immaginare due dimensioni fondamentali che riguardano un sistema di MR condiviso: lo spazio di condivisione e le tipologie di interazioni possibili tra gli utenti.
Lo spazio di condivisione può essere fondamentalmente di tre tipologie [16]:

- *Co-located*: tutti gli utenti si trovano della medesima posizione fisica.
- *Remote*: ogni utente si trova in una posizione differente.
- *Variable*: utenti collocati nella stessa posizione fisica, possono interagire con utenti remoti.

Quindi in base alla tipologia di spazio di condivisione che viene considerato vengono messe in campo tecniche e tecnologie differenti, il cui obbiettivo è però il medesimo: permettere agli utenti di avere un'esperienza condivisa di realtà mista, indipendentemente dalla posizione in cui si trovano.
Considerando invece le interazioni possibili tra gli utenti da un punto di vista temporale, esse possono essere **sincrone** o **asincrone**.
Si parla di interazioni sincrone quando si considerano utenti che partecipano alla sessione condivisa contemporaneamente, mentre per quanto riguarda le interazioni asincrone gli utenti interagiscono con la realtà condivisa in momenti diversi.

2.1.4 Perchè condivisione?

Nelle sezioni precedenti sono stati fatti un paio di esempi introduttivi sulla potenzialità della condivisione, ma è effettivamente necessaria?
Per rispondere a questa domanda è necessario contestualizzare il concetto, considerando due contesti applicativi d'esempio di un sistema di Mixed Reality: l'ambito videoludico e quello lavorativo.
Nel primo caso, una buona parte dei prodotti videoludici sul mercato per le piattaforme tradizionali presentano funzionalità di multiplayer, che consistono nel condividere l'esperienza di gioco per raggiungere determinati obbiettivi. Con l'inizio dello sviluppo di videogiochi in ambito Mixed Reality, la possibilità di poter realizzare un'esperienza multiplayer condivisa è assolutamente indispensabile e non può essere trascurata.
Per quanto riguarda l'ambito lavorativo, si può considerare ad esempio un'applicazione di MR che fornisce la possibilità a un team di architetti di effettuare un meeting, comprendendo quindi sia membri che si trovano nella stessa stanza, che membri da remoto.

Il team ha a disposizione uno spazio condiviso in cui sono presenti planimetrie e ologrammi riguardanti il progetto a cui stanno lavorando. La realizzazione di un vero e proprio meeting interattivo permette loro di avere il focus sia sullo spazio di lavoro condiviso che sullo spazio interpersonale per comunicare con gli altri membri del team, garantendo una piena coesione anche con membri che non si trovano lì fisicamente [17].

2.2 Caratteristiche dello scenario condiviso

Dopo aver definito come viene interpretato in letteratura il concetto di esperienza condivisa e aver contestualizzato questa nozione nell'ambito della Mixed Reality, ci si rende conto che gli scenari possibili sono molteplici. Essi dipendono da vari fattori, come il contesto applicativo, la locazione degli utenti, le tecnologie utilizzate e tanto altro. Per questo motivo Microsoft ha identificato sei aspetti fondamentali da considerare durante lo sviluppo di un sistema di Mixed Reality condiviso [18].
In questa sezione verranno esaminate nel dettaglio le varie problematiche su cui è necessario concentrarsi, per fornire un solido orientamento concettuale.

2.2.1 Tipologia di condivisione

In generale ci possono essere più tipologie di condivisione dell'esperienza, potrebbe esserci una presentazione da parte di un utente virtuale con cui contemporaneamente più utenti fisici possono interagire, oppure in ambito scolastico un insegnante potrebbe tenere una lezione ad utenti virtuali lavorando su materiale olografico. La complessità dell'esperienza varia in base al ruolo che l'utente ricopre all'interno del sistema e quindi ai permessi che gli vengono forniti. Vengono di conseguenza individuate tre principali categorie con cui classificare l'esperienza condivisa:

- *Presentazione:* lo stesso contenuto viene mostrato a tutti gli utenti. Esempio: Un professore che tiene una lezione e tutti gli studenti visualizzano lo stesso materiale olografico. Il professore potrebbe comunque avere a disposizione note e suggerimenti non visibili ad altri.

- *Collaborazione:* più utenti collaborano nello stesso spazio condiviso per raggiungere obbiettivi comuni. Esempio: durante una lezione di medicina viene organizzata un'attività di laboratorio per svolgere un intervento chirurgico, gli studenti si dividono in gruppi e creano attività condivise collaborative, lavorando su un modello olografico del paziente.

- *Guida:* attività con rapporto uno a uno, in cui un utente ne aiuta un altro. Esempio: professore che guida uno studente durante l'attività di laboratorio sul modello olografico.

2.2.2 Dimensione del gruppo di utenti

Una caratteristica fondamentale da determinare quando si progetta un'esperienza condivisa di Mixed Reality è sicuramente la dimensionalità del gruppo di utenti. Microsoft ha riscontrato tre classificazioni significative: un gruppo formato da due utenti è ovviamente l'unità minima per permettere un'esperienza condivisa, un gruppo fino a sei utenti viene definito come piccolo, mentre con sette o più utenti si parla di un gruppo grande.
Il numero di utenti ha un'influenza significativa sull'esperienza, tipicamente quando si verifica un primo approccio a questo ambito, la realizzazione di un'esperienza multiutente con cardinalità uno a uno è più che sufficiente, ma ovviamente con un'ottica orientata ad un'applicazione reale bisogna considerare anche la partecipazione di gruppi più numerosi.
L'aumento dei membri del gruppo impatta il sistema a livello infrastrutturale (dati e networking), a livello grafico (riproduzione degli avatar olografici nello stesso spazio, scala degli oggetti e dell'ambiente) e anche a livello sociale (l'impatto di trovarsi in una stanza con più avatar) [19].

2.2.3 Locazione degli utenti

Come già accennato nelle sezioni precedenti, un altro dettaglio importante su cui porre il focus è la locazione degli utenti. In generale un sistema di Mixed Reality può essere classificato in tre diverse categorie in base alla loro collocazione (vedi sezione 2.1.3). Il punto di forza di un sistema di Mixed Reality viene sfruttato quando si realizzano esperienze colocalizzate, in cui gli utenti si trovano tutti nello stesso ambiente fisico e interagiscono con gli ologrammi presenti. Facendo riferimento però a contesti in cui gli utenti necessitano di interagire da remoto, vanno considerate le meccaniche con cui questo può essere reso possibile, ad esempio con lo sviluppo di avatar associati che vengono renderizzati come ologrammi e hanno la possibilità di rapportarsi con il sistema.

2.2.4 Interazioni sincrone e asincrone

Un altro quesito su cui ragionare quando si vuole sviluppare un'esperienza di realtà mista condivisa riguarda le tempistiche con cui avviene la condivisione. Come è già stato introdotto nelle sezioni precedenti (vedi sezione 2.1.3) si possono considerare interazioni sincrone o asinrcone tra gli utenti che accedono al sistema. Come è stato descritto fino ad ora, un classico esempio di esperienza condivisa può riguardare due o più utenti che sono immersi nello stesso ambiente misto e interagiscono con gli ologrammi contemporaneamente e in collaborazione, questa tipologia di interazione viene definita **sincrona**. Immaginando però un caso di studio in cui è in corso lo sviluppo di un progetto da parte di un team e viene utilizzata un'applicazione di realtà mista per poter collaborare interagendo con degli ologrammi, potrebbe accadere che non sempre l'intero team ha la disponibilità di riunirsi nell'ambiente in contemporanea. Perciò sono previste anche sessioni di lavoro individuale sul progetto nelle quali possiamo facilmente immaginare la necessità di lasciare commenti o note ai collaboratori che lavoreranno in futuro sul progetto, sfruttando perciò la *persistenza* degli ologrammi (vedi sezione 1.2.2) si realizza quella che viene chiamata interazione **asincrona**, per memorizzare ad esempio un promemoria olografico nel sistema.

2.2.5 Similarità tra gli ambienti fisici

Quando si considera un'applicazione di Mixed Reality remota, in cui una parte o la totalità degli utenti non si trova nello stesso ambiente fisico ma in posizioni diverse, è necessario valutare la similarità tra gli ambienti fisici in cui essi sono collocati.
Immaginando per semplicità un'esperienza di MR remota tra due soli utenti che sono collocati in ambienti fisici differenti, sarà determinante il fatto che i due ambienti vengano classificati come **simili** oppure come **dissimili**.
Nel primo caso si parla di ambienti che presentano la stessa struttura, non devono necessariamente essere identici ma presentano similitudini riguardo a caratteristiche come le luci, suoni ambientali, mobili e dimensioni della stanza. Si basti pensare ad esempio a due sale riunioni aziendali, in cui è usuale trovare un tavolo circolare al centro della stanza con delle sedie attorno. Viceversa se due ambienti sono dissimili non presentano le stesse caratteristiche a livello di dimensionalità, mobili, luminosità e suoni ambientali. Potrebbe essere il caso di un'esperienza di MR in cui un utente si trova in una sala riunioni aziendale, mentre l'altro in un piccolo ufficio domestico.

Considerare questo aspetto è determinante a livello di esperienza utente, si basti pensare ad un contesto nel quale è necessaria una superficie piana (per esempio un tavolo) su cui lavorare, bisogna prendere in considerazione l'eventualità che in un ambiente fisico di un utente remoto essa non sia presente e adottare quindi strategie che non compromettano l'esperienza.

2.2.6 Dispositivi in uso

Un'ultima necessaria considerazione va fatta sui dispositivi hardware in utilizzo dai vari utenti. Come già accennato nelle sezioni antecedenti, gli studi sulle tecnologie e esperienze di Mixed Reality hanno come obbiettivo la realizzazione di infrastrutture capaci di astrarre l'esperienza da vincoli implementativi sia lato software che lato hardware. Con OpenXR (vedi sezione 1.3.4) è avvenuto un grande passo in avanti sotto l'aspetto della portabilità, seguendo questa metodologia di pensiero un obbiettivo nell'ambito delle applicaizioni di MR cooperative è sicuramente di permettere un'esperienza solida e coerente indipendentemente dal dispositivo in utilizzo. Viene anche presa in considerazione la progettazione di ecosistemi in cui possono essere utilizzati dispositivi non immersivi come smartphone o PC con il ruolo di osservatori, ampliando così le possibili configurazioni e caratteristiche di un'esperienza di Mixed Reality condivisa.

2.3 Stato dell'arte

Nelle sezioni precedenti è stato esaminato nel dettaglio che cosa si intende per esperienza condivisa in ambito Mixed Reality e sono stati considerati gli aspetti e le problematiche principali che riguardano la progettazione e lo sviluppo di questi sistemi. In questa sezione verranno illustrate e approfondite le tecnologie disponibili al giorno d'oggi che identificano lo stato dell'arte nell'ambito della condivisione.
Verranno analizzati sistemi che applicano la condivisione in ambienti Mixed Reality principalmente con l'obbiettivo (tramite l'utilizzo di avatar) di permettere ad utenti che si trovano in locazioni differenti di interagire nello stesso ambiente.

2.3.1 Holoportation

Con il termine **Holoportation** si identifica un sistema di comunicazione immersivo, nel quale vengono utilizzate tecnologie per la realizzazione di esperienze di Mixed Reality e sistemi allo stato dell'arte nel campo della rilevazione 3D in tempo reale [20]. Shaharam Izadi, Timo Breuer e altri membri di un team di ricercatori Microsoft, nel 2016 hanno progettato un sistema che implementa una vera e propria holoportation e hanno realizzato una demo che ne mostra le funzionalità effettive [21]. In questo sistema ci sono tre componenti fondamentali che permettono la telepresenza. Il primo consiste in un ambiente fisico dotato di un sistema di rilevazione 3D, quindi con videocamere allo stato dell'arte nell'ambito di tecnologie per il *real-time high-quality capture*. Questo ambiente denota la parte del sistema che permette il tracciamento e la ricostruzione dell'utente tramite un avatar. L'altro componente fondamentale riguarda più propriamente gli argomenti trattati in questa tesi, ovvero la visualizzazione degli avatar come ologrammi tramite visori già citati in precedenza per la Mixed Reality come Hololens 2 (Vedi sezione 1.3.1). Infine il terzo componente indispensabile è un'infrastruttura di rete che possa connettere la parte di rilevazione e modellazione 3D degli utenti con la parte di visualizzazione dei rispettivi ologrammi, deve essere quindi realizzata un'architettura di rete che possa gestire sostanziose quantità di dati e ovviamente supportare il real-time, essendo la comunicazione in tempo reale l'obbiettivo principale della holoportation.
L'obbiettivo di questa tesi è esaminare lo stato dell'arte per quanto riguarda le architetture che permettono di realizzare esperienze condivise e attuare delle sperimentazioni per valutare le diverse soluzioni possibili.

Purtroppo in questo caso gli autori dell'articolo non si concentrano particolarmente sulla parte architetturale quanto sulla sfida più ardua che hanno compiuto con questo sistema, ovvero il tracciamento dell'utente con successiva creazione di un avatar e compressione dell'enorme mole di dati generati per poter realizzare la trasmissione.
Viene comunque fatto un accenno ad un' infrastruttura in cui ogni stazione di acquisizione è considerata come un client e tramite **TCP**, in seguito all'acquisizione e compressione dei dati, procede ad inviarli real-time agli altri client attivi per la renderizzazione. Questo per quanto riguarda la comunicazione real-time, mentre invece per la funzionalità *"Living Memories"* che verrà descritta di seguito, questi pacchetti possono essere intercettati, archiviati e riprodotti da un server di registrazione intermedio.
Con questa tecnologia in fase di evoluzione viene quindi realizzata un'esperienza di Mixed Reality condivisa, in cui un utente può interagire con un avatar olografico di un altro utente che si trova in una differente locazione geografica e se gli ambienti fisici dei due utenti sono simili (vedi sezione 2.2.5) viene notevolmente incrementata l'immersività dell'esperienza. Se ad esempio è presente una sedia in entrambi gli ambienti nella stessa posizione e un utente ci si siede sopra, l'altro utente vedrà l'ologramma sedersi sulla sedia presente nel proprio ambiente e questo incapsula perfettamente il concetto di realtà mista introdotto fino ad ora, realizzando un ecosistema in cui coesistono e interagiscono tra loro entità del mondo fisico e oggetti virtuali.
Viene infine mostrata una funzionalità molto interessante identificata con il nome *"Living Memories"*, che riguarda la capacità di poter registrare una sessione di condivisione, ciò può essere considerato banale pensando al fatto che anche una normale videochiamata ha la possibilità di essere registrata, ma in questo caso le implicazioni sono differenti.
Una volta registrata la sessione, essa può essere riprodotta come ologramma tramite il visore in uso, questo implica che come tale l'esperienza possa essere vissuta nuovamente visionando se stessi e l'altro utente rappresentati da ologrammi. Essendo un ologramma, la registrazione può essere gestita come tale, quindi ad esempio essere ridotta di dimensioni e posizionata su una superficie, per essere guardata con più comodità e con un ottica differente.
Questa dinamica ci porta a pensare che in un futuro non così lontano, non solo queste tecnologie porteranno un grande beneficio a livello lavorativo, con un attenzione per l'ambiente, facendo risparmiare viaggi anche molto lunghi, ma a livello sociale permetteranno di interagire con persone molto distanti da noi e poter consolidare un vero e proprio ricordo dell'esperienza riproducibile in qualsiasi momento e con funzionalità e immersività maggiori rispetto alle normali registrazioni a cui siamo abituati.

2.3.2 Microsoft Mesh

Per quanto riguarda tecnologie che permettono la realizzazione di meeting in ambienti Mixed Reality condivisi, uno dei sistemi più recenti sviluppati da Microsoft è Mesh. **Microsoft Mesh** è stato ufficialmente presentato nel 2021 ed è stata pubblicata una clip che mostra le principali funzionalità messe a disposizione, che verranno analizzate nel dettaglio in questa sezione [22].
A livello funzionale Mesh è concettualmente simile ad Holoportation, permette infatti di realizzare sessioni di realtà mista condivisa tra utenti che si trovano in locazioni geografiche differenti o anche nello stesso luogo.
A differenza di Holoportation però il focus in Mesh riguarda pienamente gli argomenti trattati in questa tesi, infatti essa ha come obbiettivo principale quello di fornire una solida piattaforma di comunicazione e collaborazione tramite ologrammi. Di conseguenza all'interno del sistema non vengono realizzati ologrammi realistici che aderiscono all'aspetto estetico dell'utente a cui fanno riferimento (come accade in Holoportation) ma vengono utilizzati degli avatar "grezzi" che rendono molto meno onerosa la parte di campionamento e rilevazione degli utenti. Si nota infatti che in Holoportation sono state realizzate delle vere e proprie stazioni di acquisizione, dotate di videocamere e tecnologie allo stato dell'arte per il rilevamento degli utenti, mentre per quanto riguarda Mesh sia il rilevamento che la visualizzazione degli ologrammi viene fatta utilizzando un solo dispositivo abilitante (esempio: HoloLens 2).

Microsoft ha rilasciato una panoramica dei vari scenari possibili [23]:

- *Collaborazione virtuale:* Colleghi che lavorano con fusi orari differenti possono collaborare fisicamente nella stessa stanza. Un dettaglio rilevante consente nella possibilità di Mesh di integrarsi con Microsoft 365, perciò i calendari, i contenuti e il flusso di lavoro transitano naturalmente nell'ambiente di realtà mista. Questa tipologia di esperienze condivise consente di aumentare la cooperazione e la produttività di un team di dipendenti.

- *Progettazione con riconoscimento spaziale:* Mesh migliora le revisioni di progettazione 3D consentendo agli utenti di partecipare ovunque usando qualsiasi dispositivo. Viene reso possibile visualizzare e annotare i modelli 3D in tempo reale e tutto il contenuto persiste tra le sessioni di progettazione, in modo che i team possano riprendere rapidamente da dove sono rimasti.

- *Assistenza da remoto:* Mesh consente ai dipendenti di essere assistiti da remoto in caso di problematiche da colleghi più esperti. Essi possono sovrapporsi ai dati contestuali e condividere informazioni dettagliate in modo rapido ed efficace, risolvendo i problemi più velocemente.
- *Training in cooperazione:* Viene reso possibile avviare sessioni di learning sullo stesso materiale olografico condiviso. Ciò può essere indispensabile in ambiti come la chirurgia o la manutenzione delle attrezzature, in cui per imparare è necessario utilizzare le stesse risorse fisiche. Mesh aiuta quindi a migliorare l'efficacia della formazione virtuale riducendo al tempo stesso i costi di viaggio e logistica.
- *Ospitare incontri virtuali:* Viene permesso di ospitare incontri virtuali in un ambiente di Mixed Reality basato su Mesh, che migliora l'esperienza dei normali meeting in ambienti non aumentati.

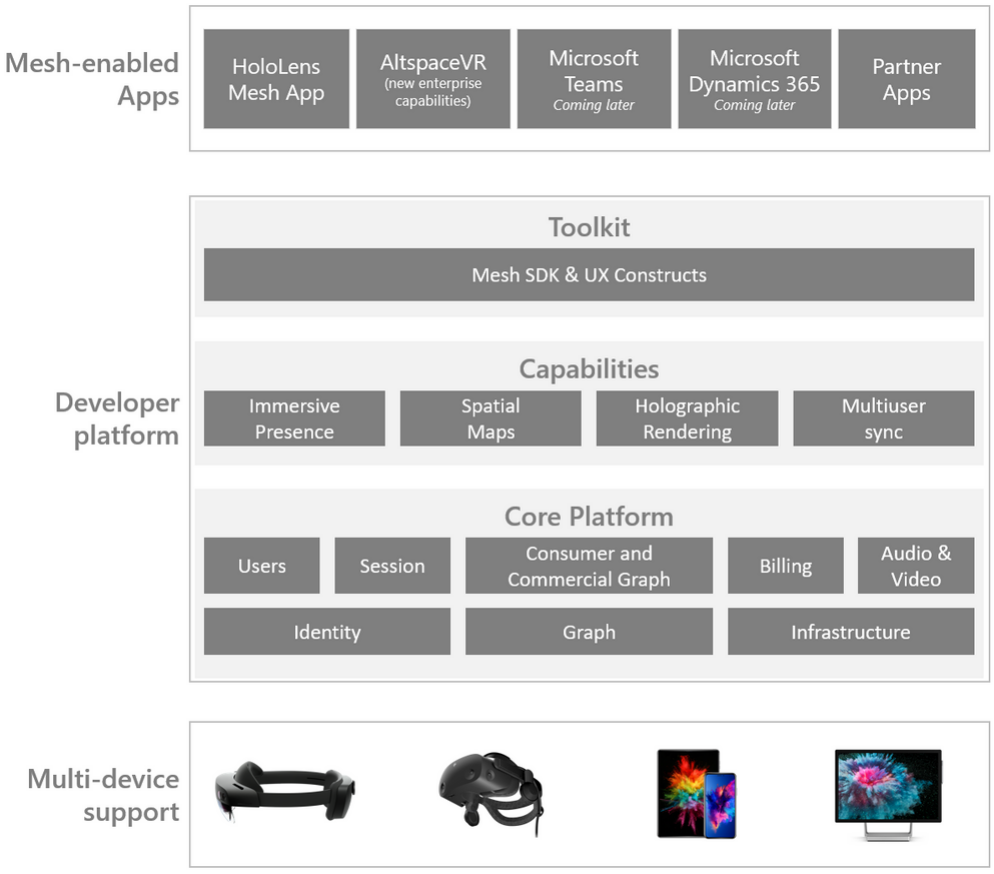

Figura 2.2: Microsoft Mesh overview.

Abbiamo notato come Mesh incapsuli formalmente molte delle funzionalità che a livello teorico dovrebbe possedere un sistema di Mixed Reality condiviso, ma a livello tecnico e architetturale come è strutturato?
Essendo una piattaforma ancora in fase di sviluppo non sono resi disponibili dettagli esaustivi sulla sua struttura ma in figura 2.2 sono riassunte le sue caratteristiche note. In generale Microsoft ha descritto Mesh anche dal punto di vista dei device e della portabilità, rappresentandolo come un sistema *"Multi-device support"* in cui l'esperienza condivisa può essere effettuata con device di diversa tipologia [24]. A livello architetturale la caratteristica fondamentale è che Mesh utilizza **Microsoft Azure** per quanto riguarda l'infrastruttura di rete, quindi a differenza dell'esempio di Holoportation in cui erano presenti client connessi tra loro tramite **TCP**, è necessaria una dipendenza dall'infrastruttura **cloud** di Microsoft.

2.3.3 JoinXR

L'utlima nonchè più recente tecnologia che verrà presentata in questo capitolo è **JoinXR**.
JoinXR è un sistema sviluppato da Fracture Reality [25], un gold partner di Microsoft specializzato nello sviluppo di piattaforme collaborative su HoloLens 2. Essendo una tecnologia presentata nel 2022, sono stati resi noti pochissimi dettagli che però sono sufficienti ai fini della tesi, per analizzare analogie e differenze rispetto ad Holoportation e Microsoft Mesh.
In generale il focus principale è sempre lo stesso: realizzare una comunicazione in un ambiente di Mixed Reality, permettendo ad utenti distanti di collaborare nello stesso ambiente con del materiale condiviso. Anche JoinXR come Mesh è concentrata su meeting tipicamente aziendali o formativi, mentre Holoportation è stata disegnata più come un mezzo per incrontrare persone distanti e memorizzare l'esperienza sotto forma di ricordo olografico.
Quindi la domanda che sorge spontanea è: qual'è la differenza rispetto alle tecnologie già presentate?
Il punto chiave di JoinXR, che la differenzia dalle due precedenti tecnologie è l'utilizzo di **Azure Remote Rendering**.

Azure Remote Rendering

Azure Remote Rendering (ARR) è un servizio che consente di eseguire il rendering di contenuto 3D interattivo di alta qualità nel cloud e trasmetterlo in streaming in tempo reale a dispositivi come ad esempio HoloLens 2 [26].
Come si può immaginare, gli HMD (Head-Mounted Display) possiedono limitate capacità di rendering e in alcuni contesti questo può risultare inaccettabile. Si basti pensare ad un'azienda che sviluppa motori per automobili, sicuramente necessiterà di prototipi olografici complessi e ad alta definizione per poterci interagire fedelmente come se fossero veri e propri prototipi fisici. Per far fronte a questo problema Azure Remote Rendering offre un servizio di rendering sul cloud, tramite il quale il carico di lavoro viene spostato su GPU di fascia alta, che una volta renderizzato il contenuto lo trasmettono in streaming come segnale video ai dispositivi.
In figura 2.3 viene presentato il risultato di una renderizzazione utilizzando un dispositivo con capacità limitate, il modello originale era composto da oltre 18 milioni di triangoli ma con una renderizzazione non appropriata il numero di triangoli viene ridotto a 200.000, rendendo il modello del motore non più aderente a quello originale.

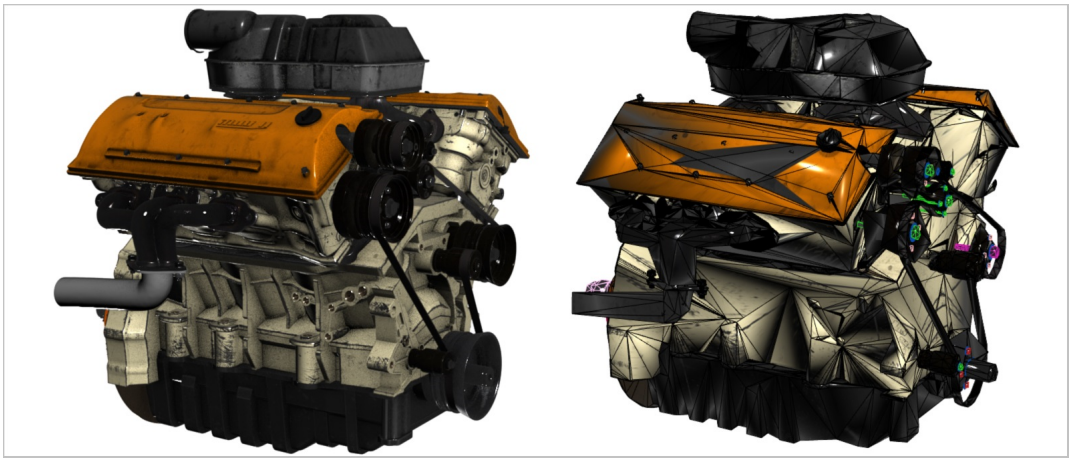

Figura 2.3: Rendering senza Azure Remote Rendering.

ARR supporta inoltre il **rendering ibrido**, normalmente in applicazioni di MR è necessario progettare anche un'interfaccia utente che in questo caso può essere sviluppata in maniera indipendente da ARR, ad esempio utilizzando MRTK (vedi sezione 1.3.3) e successivamente la renderizzazione locale dell'interfaccia verrà unita automaticamente alla renderizzazione remota creando un unico contenuto.

Infine è una problematica comune avere una situazione in cui anche una GPU di fascia alta non realizzi un rendering sufficientemente preciso, in questo caso ARR supporta il **rendering multi-GPU**, grazie al quale il carico viene distribuito su più schede e i risultati uniti in una sola immagine.

Dopo questa breve panoramica su Azure Remote Rendering, è necessario contestualizzarne l'utilizzo all'interno di JoinXR. JoinXR presenta caratteristiche comuni con Mesh e Holoportation. A livello di funzionalità aderisce molto a Microsoft Mesh, infatti Fracture Reality in una breve presentazione di JoinXR individua i seguenti punti di forza:

- Incontra istantaneamente colleghi e clienti da tutto il mondo in riunioni olografiche in tempo reale.
- JoinXR renderizza i tuoi formati di dati 3D, CAD e BIM direttamente utilizzando la più recente tecnologia di streaming cloud.
- Crea le tue presentazioni combinando documenti, immagini e video insieme a dati di progettazione 3D.

Notiamo quindi che come già accennato si parla di creazione di sessioni di MR condivise in cui possono essere integrate presentazioni con immagini documenti e video in formato olografico. ARR entra in gioco in riferimento al rendering di dati 3D, CAD e BIM in cui viene evidenziata quindi la possibilità di renderizzare prototipi molto complessi con un elevata affidabilità e precisione, rendendo la collaborazione in ambiente lavorativo estremamente efficiente. Per quanto riguarda le caratteristiche in comune con Holoportation, JoinXR sempre tramite Azure Remote Rendering presenta la possibilità di avere quelli che vengono definiti "*photoreal-avatars*" ovvero avatar aderenti alla realtà che hanno caratteristiche estetiche molto simili all'utente a cui fanno riferimento, questo si contrappone a Mesh dove gli avatar possiedono una struttura non realistica e semplificata.

Capitolo 3

Architettura in rete locale

Questa tesi si pone due obbiettivi fondamentali. Il primo è stato raggiunto nei capitoli precedenti e consiste nell'esaminare il panorama delle tecnologie allo stato dell'arte che permettono la realizzazione di esperienze di Mixed Reality condivise, descrivendole in maniera esaustiva.
In questo capitolo e nei successivi verrà posto il focus sul secondo obbiettivo, che viene identificato nella parte progettuale della tesi, ovvero considerare soluzioni architetturali indipendenti dal cloud per realizzare esperienze condivise. Nei capitoli precedenti sono state illustrate tecnologie già esistenti, più o meno complesse, che permettono questa tipologia di esperienze, ma la sfida principale di questo progetto di tesi riguarda riuscire a progettare da zero un'infrastruttura, che possa costituire la base per una tecnologia simile a quelle presentate, ma indipendente dal cloud.

3.1 Analisi concettuale

Una problematica fondamentale che è necessario affrontare riguarda l'architettura di un'applicazione di Mixed Reality condivisa.
Se si considerano applicazioni di MR che non contemplano l'esperienza condivisa si parla di applicazioni cosiddette locali, che vengono eseguite dal dispositivo in uso e non interagiscono in altro modo con l'esterno. A livello concettuale, questo implica che gli ologrammi creati a run-time sono legati ad una specifica esperienza utente ed esistono solo localmente e ciò rende l'architettura del sistema molto più semplice, ma chiaramente non predisposta alla condivisione dell'esperienza. Facendo riferimento al concetto di Mirror World [4] ci si rende conto che in futuro si prevede una pervasività e complessità di questi sistemi, tale per cui un'architettura costituita solamente da un'applicazione locale non è sufficiente. Inoltre aumentando il livello di astrazione, è necessario poter pensare di avere un vero e proprio mondo digitale persistente, che esiste ed

evolve seguendo determinati comportamenti, indipendentemente dal fatto che un utente lo stia visualizzando tramite un dispositivo. Esaminando questo concetto ci si rende perfettamente conto della profonda differenza, non solo tecnologica ma anche concettuale, tra ologrammi digitali creati a run-time su uno specifico dispositivo e un vero e proprio mondo digitale persistente, che esiste indipendentemente da quello fisico e in cui dispositivi come HoloLens 2 sono solamente tecnologie abilitanti per poter interagire con esso.
Esaminando le tecnologie presentate nei capitoli precedenti, si evince un dettaglio comune molto importante per quanto riguarda la loro architettura: sono tutte dipendenti dal servizio cloud di Microsoft.
Questo ci porta ad una domanda che rappresenta il fulcro del progetto di tesi: è possibile realizzare un architettura che permetta un'esperienza di MR condivisa senza utilizzare il cloud? e che prestazioni offre?

3.2 Struttura architetturale

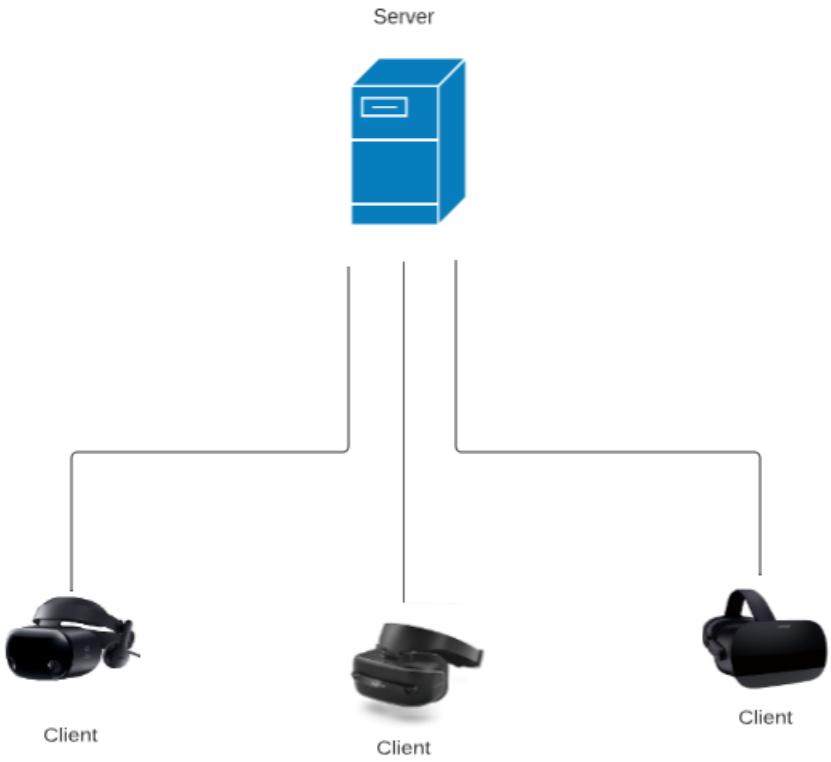

Figura 3.1: Struttura architetturale client-server.

Per poter realizzare un'infrastruttura indipendente dal cloud Microsoft è necessario considerare tecnologie e protocolli comuni a livello di rete. In generale si è pensato di realizzare l'infrastruttura in modo più semplice e chiaro possibile, senza utilizzare tecnologie o protocolli che ne compromettessero la complessità. Una prima considerazione va fatta sul concetto di mondo digitale persistente che continua ad esistere ed evolvere indipendentemente dal fatto che un utente stia interagendo con esso. Per realizzare questa specifica una tecnologia che permette questo tipo di funzionalità è sicuramente un server.
Possiamo quindi considerare una semplice architettura client-server (figura 3.1) in cui tutti i dispositivi abilitanti (HMD) che permettono di interagire con gli ologrammi sono identificati come client e comunicano attraverso la rete con un server. Il server ha quindi lo scopo di mantenere al suo interno il mondo digitale e fornirne lo stato ai client connessi ad esso, in modo che tutti possano offrire la stessa esperienza utente.
Questa tipologia di approccio può essere definita come *"fog-oriented"*, dato che si considera l'infrastruttura di rete più semplice possibile, con il server situato in rete locale. Questa scelta è stata fatta per poter semplificare del tutto la parte di interconnessione e rimuovere le dipendenze da infrastrutture cloud.
A questo punto continuando ad analizzare l'architettura si evince la mancanza di un dettaglio fondamentale: il protocollo di comunicazione.
Anche in questo caso è stata fatta una scelta che permette di sviluppare la sperimentazione nel modo più chiaro e semplice possibile, quindi sono stati selezionati i due protocolli classici che si trovano a livello *Transport* dello stack di rete ovvero **TCP** e **UDP**.
Riassumendo ci troviamo quindi davanti ad un'architettura client-server, che prevede una rete locale con all'interno un server principale, che contiene lo stato del mondo digitale e quindi degli ologrammi e tramite TCP oppure UDP comunica con gli HMD (ad esempio HoloLens 2) che svolgono il ruolo di client e visualizzano l'ambiente olografico condiviso.
Per quanto riguarda la logica di comunicazione tra client e server, essa può essere realizzata con varie strategie. Per favorire la semplicità e compattezza della soluzione si è deciso di utilizzare una tecnica di **polling** tramite la quale vengono costantemente richieste al server le informazioni di aggiornamento, effettuando dei miglioramenti architetturali sarebbe possibile realizzare una comunicazione **event-driven** che risulterebbe sicuramente più efficiente ma di conseguenza più elaborata.
Questa architettura presenta una scalabilità che dipende dalle tecnologie selezionate per l'implementazione del server, ma in generale è chiaro che non si possa comparare con un'applicazione non sperimentale, per la quale sarebbe sicuramente necessario spostare il server all'esterno della rete locale, o comunque renderlo accessibile anche da altre reti.

3.3 Pattern di progettazione

Una volta definita nel dettaglio la parte architetturale è necessario fare chiarezza sull'organizzazione a livello progettuale dei vari componenti del sistema. A fronte dell'esperienza in materia di Programmazione ad Oggetti e Ingegneria del Software per coordinare e organizzare in maniera coerente l'applicazione è possibile mettere in pratica un pattern di progettazione. In questo caso un pattern che aderisce alle esigenze del progetto è sicuramente il pattern **Model View Controller (MVC)** [27].
Risulta sicuramente naturale identificare come *Model* del sistema il mondo digitale, in quanto presenta entità (rappresentate graficamente da ologrammi) che hanno un comportamento che evolve nel tempo seguendo una determinata logica. Le entità appartenenti al Model avranno una controparte grafica, in questo caso un ologramma, che rappresenta la parte di *View* del sistema e permette all'utente di interagire con esso. Infine per coordinare l'interazione tra Model e View si possono adottare varie strategie per identificare il *Controller*, la struttura di controllo più semplice potrebbe essere un *loop* che controlla l'evoluzione del mondo digitale e quindi del Model e ne comunica i cambiamenti alla parte di View, oppure potrebbe essere un Thread separato che viene eseguito concorrentemente.
Effettuando un'unione tra l'architettura di rete precedentemente descritta e il pattern MVC otteniamo quindi un sistema in cui: lato server viene realizzata la parte di Model che comprende tutte le entità che saranno condivise durante l'esperienza e la parte di Controller che si occupa di gestire i cambiamenti delle entità e comunicare con l'esterno. Mentre lato client viene collocata la parte di View del sistema, che interagendo con il Controller tramite un apposito protocollo di rete, aggiorna gli ologrammi conseguentemente ad eventuali cambi di stato (figura 3.2).

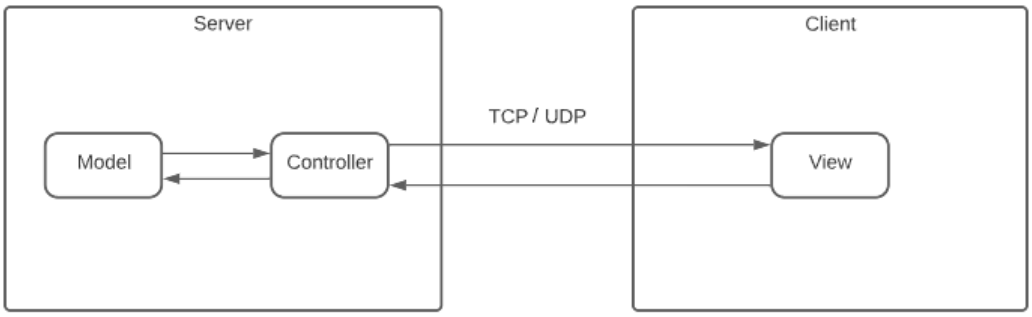

Figura 3.2: Pattern MVC applicato alla struttura client-server.

3.4 Contesto applicativo

Una volta definita con chiarezza l'architettura del sistema e il pattern di progettazione, è necessario contestualizzarli in un'applicazione sperimentale. Sia la struttura di rete che il pattern MVC rappresentano comunque soluzioni architetturali applicabili a una vasta gamma di contesti applicativi, facendo però sempre riferimento ad ambienti e applicazioni semplici, per situazioni più complesse è sicuramente necessario estendere l'architettura sotto vari aspetti.
Come è stato definito nelle sezioni precedenti, l'obbiettivo è riuscire a progettare un'architettura indipendente dal cloud per realizzare esperienze condivise di Mixed Reality e verificare le prestazioni delle possibili soluzioni.
Perciò è necessario modellare un contesto applicativo che possa permettere di raggiungere questo obbiettivo senza sviluppare più del dovuto aspetti non rilevanti in questo caso di studio.
Come applicativo di riferimento è stata definita un'esperienza nella quale è presente un semplice ologramma rappresentante un cubo che ruota rispetto ad uno dei suoi assi cartesiani. Il cubo sarà quindi parte del mondo digitale e cioè del Model dell'applicazione, la sua esistenza e il suo comportamento saranno definiti e controllati lato server e i suoi cambiamenti (in questo caso la rotazione) sarà notificata alla parte di View lato client tramite TCP o UDP.
In questo modo, tutti i client che si collegheranno al server visualizzeranno il cubo ruotare in maniera analoga e a differenza delle normali applicazioni locali, il cubo sarà un'entità unica e persistente situata in una precisa locazione (server) la cui visualizzazione potrà avvenire contemporaneamente per più utenti.
Sviluppando un'applicazione sperimentale nel contesto applicativo appena descritto, è possibile porre il focus sulla parte di modellazione dell'architettura e verifica delle prestazioni, senza dover spendere risorse nella modellazione di ologrammi complessi sia a livello grafico che a livello comportamentale.
Per testare i limiti del sistema può essere eventualmente possibile generare più ologrammi rappresentanti dei cubi, creando così una condizione di stress tramite la quale è possibile verificare le prestazioni dell'applicazione, a confronto con quelle basate su cloud.
Seguendo questa linea di modellazione verrà quindi realizzata un'applicazione molto semplice che però, una volta testati e consolidati i suoi limiti, è sicuramente estendibile. L'aspetto più interessante per un'estensione sarebbe permettere agli utenti non solo di visualizzare gli ologrammi condivisi, ma anche poterli modificare, in modo da realizzare una comunicazione bidirezionale client-server che aggiunge complessità al sistema.

Capitolo 4

Sviluppo del progetto

In questo capitolo verrà illustrata e descritta dettagliatamente la fase di implementazione del progetto di tesi, che segue la parte di modellazione dell'architettura appena progettata. Verranno esplorate le tecnologie utilizzate, alcune sono già state descritte nelle sezioni precedenti mentre altre necessitano solamente di un breve accenno.
Inizialmente verranno illustrate le parti che compongono il core dell'applicazione ovvero la struttura del Model, del Controller e della View.
Successivamente lo sviluppo è stato ramificato in due direzioni, la prima realizza un'architettura basata su TCP, che come è noto sfrutta il principio *connection-oriented* e ha l'obbiettivo di garantire la ricezione dei messaggi, mentre l'altra si basa su UDP ovvero un protocollo *connectionless* non affidabile che ha però il vantaggio della semplicità strutturale [28].
A livello architetturale le due implementazioni sono molto simili, perciò verrà illustrata prima quella basata su TCP e successivamente verranno evidenziate le differenze con quella basata su UDP.

4.1 Core dell'applicazione

In questa sezione viene descritta la struttura centrale dell'applicazione, che prescinde dal protocollo di comunicazione utilizzato. Segue quindi una prima breve panoramica sulle tecnologie selezionate per l'implementazione:

- **Server**: come linguaggio di programmazione è stato scelto **Java**, in particolare è stato utilizzato il framework **Vert.x** per poter realizzare il back-end.

- **Client**: risiede nell'applicativo di Unity quindi è stato realizzato utilizzando il linguaggio di scripting conseguente ovvero **C#**.

Cercando di costruire un quadro completo, avremo quindi un applicativo distribuito su più sistemi e organizzato come segue.
Sarà sviluppato un server Java che avrà al suo interno il Model e il Controller, esso gestirà l'evoluzione del mondo digitale e la comunicazione con i client attraverso la rete locale.
Sarà presente un client all'interno di un'applicazione Unity, di cui verrà fatto il deploy su un Head Mounted Display e tramite lo script in C# si collegherà al server.
Una volta stabilita la connessione tra client e server l'HMD riceverà attraverso la rete locale dei pacchetti TCP o UDP dal server, che lo informeranno sullo stato delle entità del model. Di conseguenza verranno aggiornati i componenti della parte di View (gli ologrammi) in modo da sincronizzarli con lo stato delle rispettive entità presenti nel server. In questo modo, si rispetta il requisito del problema di mantenere indipendente la progressione del mondo digitale dal fatto che un utente lo stia visualizzando: finchè il server è in esecuzione il cubo ruota e nel momento in cui un visore viene connesso ad esso, il cubo verrà visualizzato nella posizione in cui si trova in quel momento.

4.1.1 Model

Come già anticipato il Model è situato lato server, quindi sviluppato in **Java**. Esso consiste nel mondo digitale persistente e contiene quindi le entità logiche dell'applicazione. Essendo l'applicativo sperimentale molto semplice, la parte essenziale di esso è formata dalla classe `Cube` rappresentante un cubo con logica *Object-Oriented*, che avrà poi una corrispondenza con l'ologramma che ne permetterà la visualizzazione.
Come si può notare dal listato 4.1 la classe Cube contiene al suo interno tre attributi fondamentali: `position`, `rotation` e `scale`. Questi tre attributi sono oggetti che fanno riferimento alle omonime classi di utility, che sono state sviluppate per poter mantenere le coordinate dei tre assi cartesiani x, y e z per la posizione, la rotazione e la scala.
Per lo sviluppo di questo progetto sono state individuate queste tre semplici proprietà, che permettono di effettuare uno studio completo senza aggiungere complessità. Una possibile espansione del progetto potrebbe ampliare la classe del cubo considerando anche proprietà più elaborate, come ad esempio il colore delle facce. Gli attributi sono tutti privati e accessibili tramite getter e setter pubblici, ed è infine presente un costruttore per poter inizializzare il cubo. Come è intuibile questa classe verrà utilizzata nel server per creare nuovi cubi all'interno del mondo digitale e gestirne l'evoluzione nel tempo.
Considerando un'estensione dell'applicativo sarà necessario creare nuove classi, che verranno associate ad ologrammi eventualmente più complessi ed elaborati.

```
package model;

public class Cube {

   private Position position;
   private Rotation rotation;
   private Scale scale;

   public Cube(Position position, Rotation rotation, Scale scale) {
      this.position = position;
      this.rotation = rotation;
      this.scale = scale;
   }

   public Position getPosition() {
      return position;
   }

   public void setPosition(Position position) {
      this.position = position;
   }

   public Rotation getRotation() {
      return rotation;
   }

   public void setRotation(Rotation rotation) {
      this.rotation = rotation;
   }

   public Scale getScale() {
      return scale;
   }

   public void setScale(Scale scale) {
      this.scale = scale;
   }
}
```

Listato 4.1: Classe rappresentante il cubo in Java

4.1.2 Controller

Il Controller rappresenta il cuore dell'applicazione ed ha lo scopo di conciliare la parte logica con la parte grafica per tutta la durata dell'esecuzione dell'applicativo. Facendo riferimento al listato 4.2 , si nota che il controller è rappresentato dalla classe `RunService` che come intuibile, contiene un main e rappresenta l'entry-point lato server.
Avendo sviluppato la parte back-end con logica Object-Oriented il server (TCP in questo esempio) è rappresentato da un oggetto di tipo `TCPServer` che verrà esaminato nel dettaglio nelle sezioni successive.
Ad ogni modo una volta creato il server e averlo messo in ascolto tramite il metodo `start()` si nota la vera e propria struttura di controllo.
Per mantenere i componenti dell'applicativo semplici e comprensibili, è stata scelta la struttura base che è sufficiente al controller per poter raggiungere gli obbiettivi previsti: un loop.
All'interno di esso tramite il metodo `send()` del server TCP si procede ad inviare ai client connessi le informazioni di aggiornamento riguardo al mondo digitale. Viene utilizzata una sleep per poter scandire l'invio delle informazioni, di conseguenza la tempistica di tale funzione va correttamente tarata in base alle esigenze e alle prestazioni che può raggiungere il sistema.
In una fase evolutiva dell'applicativo, può essere permesso ai client di inviare informazioni al server per effettuare la modifica delle entità del model. In tal caso nel loop di controllo verrà inserita una funzione per la ricezione oltre a quella già presente per l'invio delle informazioni. Infine il loop di controllo può essere sostituito da un Thread separato che viene eseguito in background ed è eventualmente strutturato con una logica ad eventi.

```
public class RunService {

   public static void main(String[] args) throws Exception{

      TCPServer tcpServer = new TCPServer();
      tcpServer.start();

      while(true) {
         tcpServer.send();
         Thread.sleep(100);
      }
   }
}
```

Listato 4.2: Controller contenente il main e il loop di controllo

4.1.3 View

L'ultimo componente fondamentale dell'architettura è la View.
Essa incapsula, come si evince dal nome, la parte prettamente grafica del sistema. Come è stato già accennato questa parte di architettura è identificata da un applicativo sviluppato in Unity, di cui viene successivamente effettuato il deploy su un Head Mounted Display.
La parte di View ha quindi lo scopo di associare un'entità grafica alla corrispettiva entità logica del mondo digitale situata nel Model. In questo caso di studio perciò, l'entità grafica consiste in un ologramma rappresentante il cubo di cui è stata precedentemente definita la classe in Java lato server.
A livello concettuale, la parte di View non dovrebbe contenere entità al suo interno prima di stabilire una comunicazione con il Controller, in quanto il fulcro del mondo digitale risiede lato server e la parte grafica deve dipendere da esso per la visualizzazione la creazione o la modifica degli ologrammi. Perciò il cubo mostrato in figura 4.1 non è un semplice `GameObject` che viene aggiunto manualmente alla scena, ma è la rappresentazione grafica di un'entità persistente presente lato server, ed ha di conseguenza un comportamento sincronizzato con essa.
Per la realizzazione del progetto Unity sono state utilizzate le tecnologie presentate nella sezione 1.3, in particolare MRTK ha facilitato lo sviluppo dell'applicativo, per quanto esso non contenga componenti particolarmente elaborati.
Nelle prossime sezioni verranno esaminate nel dettaglio le dinamiche e le strategie adottate per realizzare la corretta interazione tra Model, View e Controller.

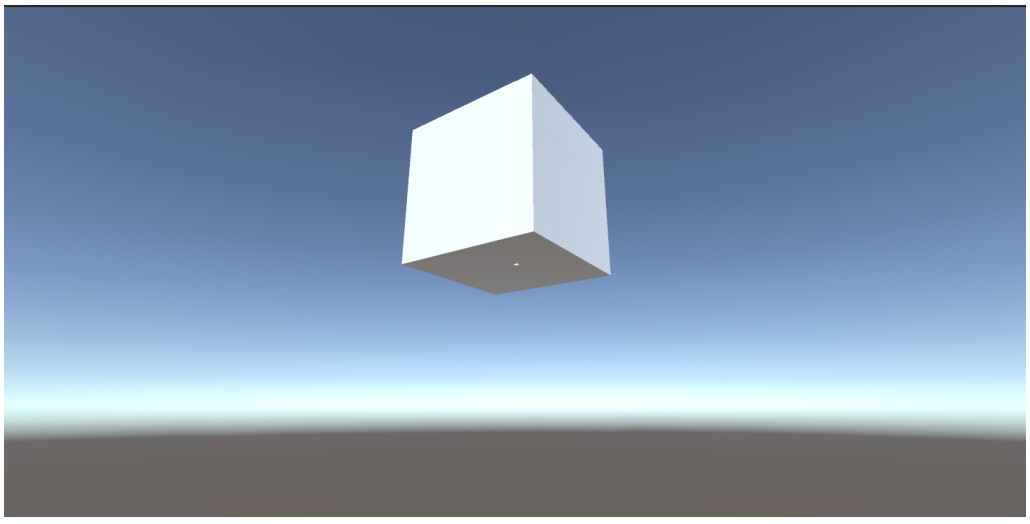

Figura 4.1: View visualizzata tramite l'emulatore di Unity.

4.2 Strategia basata su protocollo TCP

Dopo aver ampiamente descritto il fulcro dell'applicazione è necessario esaminare le due diverse strategie che sono state attuate a livello di rete.
Come già anticipato la struttura e collocazione di Model, View e Controller e le tecnologie utilizzate sono simili in entrambe le implementazioni, la differenza fondamentale risiede nel protocollo di rete utilizzato per la comunicazione.
Come è noto sia TCP che UDP presentano punti di forza e debolezze che li differenziano e li rendono adattabili a diverse situazioni ed esigenze. In questa sezione verrà mostrata la soluzione implementativa che utilizza TCP, mentre nella sezione successiva sarà possibile esaminare una soluzione basata sul protocollo UDP.

4.2.1 Server

Come già anticipato il server TCP è stato implementato in Java.
In particolare è stato utilizzato il framework **Vert.x** [29].
Vert.x permette con facilità di sviluppare piattaforme e servizi web con strategie reactive, in questo caso di studio ovviamente non sono state sfruttate a pieno le sue funzionalità, in quanto era necessario realizzare una parte di back-end semplice e compatta.
Facendo riferimento al listato 4.3 che mostra l'implementazione effettiva del server, si noti che esso è rappresentato dalla classe `TCPServer`, di cui è stato istanziato e utilizzato un oggetto all'interno del Controller (vedi sezione 4.1.2).
Come già esplicato più volte, il server ha il compito di creare e mantenere la struttura del mondo digitale, applicando questo concetto al nostro caso di studio esso si traduce nell'obbiettivo di avere un riferimento ad una o più entità di tipo `Cube`. Nella prima fase di sperimentazione è sufficiente un solo cubo, ma con un'ottica volta all'espansione dell'applicativo, si considera la possibilità di generare più cubi all'interno del mondo digitale, perciò viene utilizzata la lista `cubes` per memorizzarne i riferimenti.
Procedendo ora ad esaminare nel dettaglio le parti salienti dell'implementazione del server contenuta nell listato 4.3, si noti che la classe `TCPServer` è identificata da Vert.x come un `AbstractVerticle` [30], perciò è necessario estendere l'omonima classe astratta che richiede di implementare il metodo `start()` per l'inizializzazione effettiva del server. Di seguito verranno descritte in dettaglio le strategie implementative adottate nella realizzazione della soluzione.

```
public class TCPServerBase extends AbstractVerticle {
   private NetSocket serverNetSocket;
   private List<Cube> cubes = new LinkedList<>();
   private boolean isConnected = false;
    private long receiveTimestamp;
   private List<Long> allSendTimestamp = new LinkedList<>();
   private static float DELTA_ROTATION = 2.0f;
   private static int NUMBER_OF_CUBES = 10;
   private static float SCALE = 0.2f;

   @Override
    public void start() throws Exception {
      this.initCubes();
      Vertx vertx = Vertx.vertx();
        NetServer server = vertx.createNetServer();

        server.connectHandler(new Handler<NetSocket>() {

            @Override
            public void handle(NetSocket netSocket) {
                System.out.println("Incoming connection!");

                netSocket.handler(new Handler<Buffer>() {

                    @Override
                    public void handle(Buffer inBuffer) {
                        System.out.println("Incoming data: " +
                            inBuffer.length());

                        String data = inBuffer.getString(0,
                            inBuffer.length());

                        System.out.println("Data: " + data);
                        Date date = new Date();
                     receiveTimestamp = date.getTime();

                     long sendTimeStamp = allSendTimestamp.get(0);

                     long delay = (receiveTimestamp-sendTimeStamp)/2;

                     allSendTimestamp.remove(0);

                     System.out.println("Send timestamp: " +
```

```
                    sendTimeStamp + " ms");
                System.out.println("Receive timestamp: " +
                    receiveTimestamp + " ms");
                System.out.println("Delay: "+ delay + " ms");
               }
            });
            serverNetSocket = netSocket;

            JsonArray packetJson = new JsonArray();

            for(Cube cube : cubes) {
           JsonObject cubeJson = new JsonObject();
           cubeJson.put("cubeID", cube.getId());
           cubeJson.put("position",
               cube.getPosition().toString());
           cubeJson.put("rotation",
               cube.getRotation().toString());
           cubeJson.put("scale", cube.getScale().toString());
           packetJson.add(cubeJson);
        }

            Buffer outBuffer = Buffer.buffer();
            outBuffer.appendString("{\"Items\":" +
                packetJson.toString() + "}");
            serverNetSocket.write(outBuffer);

            System.out.println("[TCP] Sending setup packet to
                clients");

            try {
           Thread.sleep(100);
           isConnected = true;
        } catch (InterruptedException e) {
           e.printStackTrace();
        }
        }
    });
    server.listen(10000);
    System.out.println("Server TCP listening on port 10000...");
  }
}
```

Listato 4.3: Implementazione del metodo start() del server TCP.

Prima di continuare nell'analisi del codice è necessario fare una breve panoramica sulle variabili presenti all'interno della classe:

- `serverNetSocket`: variabile che mantiene il riferimento alla socket per la connesione TCP.
- `cubes`: lista degli oggetti appartenenti al model, in questo caso istanze della classe `Cube`.
- `isConnected`: variabile booleana utilizzata per verificare se è avvenuta una connesione al server.
- `DELTA_ROTATION`: costante per indicare il delta di rotazione dei cubi e regolarne di conseguenza la velocità.
- `NUMBER_OF_CUBES` : costante che stabilisce quanti cubi generare.
- `SCALE`: costante che identifica la dimensione dei cubi lungo i tre assi cartesiani.
- `receiveTimestamp` : variabile che memorizza il tempo in millisecondi di arrivo di un acknowledge.
- `allSendTimestamp` : lista in cui vengono inseriti i tempi di invio dei messaggi in millisecondi.

All'interno del metodo `start()` viene creata un'istanza di un oggetto `Vertx`, che a sua volta viene utilizzato per creare un'istanza del server. Ad esso viene connesso un `Handler` che ha lo scopo di fornire una funzione di callback, che verrà eseguita quando un client effettuerà la connessione ad esso.
A questo punto è necessario descrivere la parte centrale che garantisce il funzionamento del sistema.
Inanzitutto il server viene messo in ascolto sulla rete locale all'indirizzo dell'elaboratore **192.168.40.100** con numero di porta **10000**. Una volta in ascolto, appena riceve una connessione, il server invia un primo **pacchetto di setup** che ha la funzione di informare il client appena connesso su quanti cubi sono presenti all'interno del mondo digitale, la loro posizione, la loro dimensione e un'eventuale rotazione. Queste caratteristiche iniziali possono essere modificate cambiando i valori delle costanti precedentemente descritte, avendo così la possibilità di creare vari scenari.

Per la composizione del pacchetto è stato selezionato il formato **JSON** [31] in quanto presenta una struttura e una compattezza che rendono più facile la coordinazione e l'interpretazione dei messaggi ed è inoltre direttamente supportato da Vertx.
Il pacchetto consiste quindi in un array JSON, in cui ogni elemento rappresenta un cubo e contiene al suo interno (oltre all'identificativo univoco dell'oggetto) i valori relativi alle sue proprietà.

```
private void initCubes() {
    for(int i= 0; i < NUMBER_OF_CUBES; i++) {
        Position position = new Position(0,0,-(float)i);
        Rotation rotation = new Rotation(0,0,0);
        Scale scale = new Scale(SCALE, SCALE, SCALE);
        cubes.add(new Cube(i, position, rotation, scale));
    }
}
```

Listato 4.4: Funzione di inizializzazione dei cubi.

Si noti che all'interno del metodo `start()` viene invocata la funzione `initCubes()`, la sua implementazione è mostrata in dettaglio all'interno del listato 4.4. Essa consiste in una creazione delle entità del model (basata sulla costante `NUMBER_OF_CUBES`) che popola il mondo digitale riempiendo la lista `cubes`. Una volta creato il mondo digitale, è necessario progettare una funzione che venga invocata periodicamente e lo faccia evolvere secondo una determinata logica.

```
private void rotateCubes() {
    for(int i=0; i<cubes.size(); i++) {
        Rotation oldRotation = cubes.get(i).getRotation();
        Rotation newRotation;
        if(i%2 == 0) {
            newRotation = new Rotation (oldRotation.getX()+
            DELTA_ROTATION,oldRotation.getY(),oldRotation.getZ());
        }else {
            newRotation = new Rotation
                (oldRotation.getX(),oldRotation.getY()+
            DELTA_ROTATION,oldRotation.getZ());
        }
        cubes.get(i).setRotation(newRotation);
    }
}
```

Listato 4.5: Funzione che effettua l'evoluzione del mondo digitale.

La funzione `rotateCubes()` ha lo scopo di far evolvere il mondo digitale, in questo caso applicando una rotazione sull'asse x ai cubi di indice pari e una rotazione sull'asse y ai cubi di indice dispari.
Procedendo nel descrivere il funzionamento della comunicazione, in seguito all'invio del primo pacchetto di setup che avviene nella parte finale della funzione `start()`, è possibile proseguire con l'invio dei pacchetti di aggiornamento. Si noti dal listato 4.2 che nel loop di controllo viene periodicamente invocata la funzione `send()` del server TCP, essa ha lo scopo di inviare ai client connessi gli aggiornamenti riguardanti il Model, per permetterne la renderizzazione lato View.

```
public void send() {
    if(!isConnected) {
        return;
    }

    this.rotateCubes();
    System.out.println("[TCP] Sending update packet to clients");

    JsonArray packetJson = new JsonArray();

    for(Cube cube : cubes) {
        JsonObject cubeJson = new JsonObject();
        cubeJson.put("cubeID", cube.getId());
        cubeJson.put("position", cube.getPosition().toString());
        cubeJson.put("rotation", cube.getRotation().toString());
        cubeJson.put("scale", cube.getScale().toString());
        packetJson.add(cubeJson);
    }

    Buffer outBuffer = Buffer.buffer();
    outBuffer.appendString("{\"Items\":" + packetJson.toString() +
        "}");
    serverNetSocket.write(outBuffer);

    Date date = new Date();
    allSendTimestamp.add(date.getTime());
}
```

Listato 4.6: Metodo per l'invio degli aggiornamenti ai client tramite TCP.

Essa non fa altro che far evolvere il mondo digitale ruotando tutti i cubi memorizzati nella lista richiamando il metodo `rotateCubes()`, e successivamente compone un pacchetto in formato JSON che ha lo scopo di comunicare i cambiamenti di stato avvenuti nel model.
Infine, una volta strutturato il pacchetto, viene utilizzata la socket TCP per effettuarne l'invio al client connesso.
L'ultima dinamica che necessita un approfondimento riguarda il sistema di misurazione delle prestazioni. Nelle fasi successive dell'esplorazione, sarà necessario avere un riscontro matematico delle prestazioni di questa architettura, quindi è indispensabile predisporla a questo tipo di verifiche.
Considerando un'architettura con un solo client, tutte le volte che viene inviato un pacchetto di update dal server, tramite la classe `Date` viene rilevato il tempo corrente in millisecondi e questa misurazione viene aggiunta alla lista `allSendTimestamp`. Una volta inviato il pacchetto di update, il client effettua la ricezione, interpreta il messaggio e applica le renderizzazioni necessarie, infine invia un pacchetto di **acnkowledge** al server.
In questo modo **l'handler asincrono** lato server al momento della ricezione dell'ack, consuma il primo timestamp presente nella lista `allSendTimestamp` (vedi listato 4.3) e lo utilizza per fare i calcoli necessari che saranno approfonditi nelle sezioni successive.

4.2.2 Client

Per quanto riguarda il client TCP, esso consiste in uno script all'interno dell'applicativo Unity, di conseguenza è stato sviluppato in C#.
In questa sezione verrà illustrato nel dettaglio il suo funzionamento, esaminandone le parti più rilevanti.
Innanzitutto si ricorda che il client è stato progettato per essere eseguito nel HMD successivamente alla fase di deploy. Di seguito vengono mostrate le funzioni che si occupano della connessione con il server TCP e della ricezione dei messaggi.

```
private void ConnectToTcpServer ()
{
    try {
        clientReceiveThread = new Thread (new
            ThreadStart(ListenForData));
        clientReceiveThread.IsBackground = true;
        clientReceiveThread.Start();
    }
    catch (Exception e) {
        NewLog("On client connect exception " + e);
    }
}

private void ListenForData()
{
    try {
        socketConnection = new TcpClient("192.168.40.100", 10000);
        Byte[] bytes = new Byte[1024];
        while (true) {
            using (NetworkStream stream =
                socketConnection.GetStream()) {
                int length;

                while ((length = stream.Read(bytes, 0, bytes.Length))
                    != 0) {
                    var incommingData = new byte[length];
                    Array.Copy(bytes, 0, incommingData, 0, length);

                    string msgString =
                        Encoding.ASCII.GetString(incommingData);
                    int count = Regex.Matches(msgString,
                        "Items").Count;
```

```
                    if(count != 1){
                        continue;
                    }

                    if(firstMsg){
                        setupMessages = JsonHelper.
                                    FromJson<UpdateMessage>(msgString);
                        create = true;
                        firstMsg = false;
                    }else{
                        update = true;
                        queue.Enqueue(JsonHelper.
                                    FromJson<UpdateMessage>(msgString));
                    }
                }
            }
        }
        catch (SocketException socketException) {
            NewLog("Socket exception: " + socketException);
        }
}
```

Listato 4.7: Implementazione della connesione al server TCP in C#.

Come mostrato nel listato 4.4, sono necessarie due funzioni per la connessione con il server, il primo metodo è `ConnectToTcpServer()` che crea un **thread** dedicato per la comunicazione e gli associa la funzione `ListenForData()` che ha lo scopo di ricevere i messaggi inviati al client. All'interno di tale funzione viene utilizzata una **socket** configurata con l'indirizzo e la porta del server TCP e tramite essa vengono ricevuti i messaggi.
Il thread del client possiede una struttura di controllo a loop simile a quella già mostrata lato server, con la differenza che tramite un'istanza di `NetworkStream` riceve i pacchetti anzichè inviarli. Quando viene ricevuto un nuovo pacchetto, viene controllato se esso è il primo della comunicazione, perchè in tal caso si tratta di un **pacchetto di setup** con il quale il server definisce quanti cubi sono presenti nel mondo digitale, le loro dimensioni e la loro posizione. Se il pacchetto ricevuto non è il primo, allora significa che è un **pacchetto di update** con il quale il server comunica gli aggiornamenti riguardanti l'evoluzione del mondo digitale. Viene utilizzata una **coda** all'interno del quale vengono inseriti i messaggi ricevuti e come verrà mostrato successivamente, ad ogni iterazione del loop del client essi vengono rimossi da essa e processati.

```
[System.Serializable]
public class UpdateMessage
{
   public int cubeID;
   public string position;
    public string rotation;
   public string scale;

    public static UpdateMessage CreateFromJSON(string jsonString)
    {
        return JsonUtility.FromJson<UpdateMessage>(jsonString);
    }
}
```

Listato 4.8: Implementazione della classe riguardante i messaggi.

Come appena accennato, ci sono due tipologie di messaggi che possono essere ricevuti: un messaggio di setup o un messaggio di aggiornamento. Lato client è quindi stata creata una classe dedicata che rappresenta un messaggio e sfruttando il paradigma ad oggetti converte automaticamente (tramite una funzione presente nella libreria `JsonUtility`) il messaggio ricevuto in formato JSON, in un oggetto di tipo UpdateMessage che ha come attributi i parametri del pacchetto. Questo rende molto semplice accedere al contenuto dei pacchetti ricevuti, istanziando un oggetto ti tipo `UpdateMessage` e accedendo alle sue proprietà.

Essendo il client uno script Unity, esso possiede i classici metodi `Start()` e `Update()` mostrati nel listato 4.9, che vengono utilizzati rispettivamente per l'inizializzazione e l'aggiornamento dei vari componenti. Nel metodo `Start()` l'unica procedura rilevante riguarda la realizzazione della connessione al server TCP con la funzione descritta precedentemente. Per quanto riguarda la funzione `Update()` essa viene invocata periodicamente per aggiornare le componenti grafiche dell'applicazione, ovvero gli ologrammi. Al suo interno in questo caso, tramite variabili booleane settate in base ai messaggi ricevuti dal server, può avvenire la creazione iniziale degli ologrammi dei cubi, oppure l'aggiornamento delle loro proprietà, coerentemente con i cambi di stato ricevuti tramite la socket. In particolare vengono rimossi i messaggi contenuti nella coda, e dato che ognuno di essi fa riferimento ad un cubo, viene controllato l'ID dell'ologramma e aggiornato coerentemente con le informazioni presenti nel pacchetto riguardanti la posizione, la rotazione e la scala. Succesivamente alla renderizzazione viene invocata la funzione `SendAcknowledge()` che si occupa di inviare un ack al server, il quale registrerà i tempi ed effettuerà i calcoli sulle performance dell'architettura che saranno approfonditi nelle sezioni successive.

```
void Start()
{
    debugGameObj = GameObject.Find("DebugTxt");
    debugConsole = debugGameObj.GetComponent<TextMeshProUGUI>();
    ConnectToTcpServer();
}

void Update()
{
    if(newLog){
        debugConsole.text = logMsg;
        newLog = false;
    }

    if(create)
    {
        for(int i = 0; i < setupMessages.Length; i++){
            GameObject newCube =
                GameObject.CreatePrimitive(PrimitiveType.Cube);
            string [] positionXYZ =
                setupMessages[i].position.Split('|');
            string [] scaleXYZ = setupMessages[i].scale.Split('|');
            newCube.transform.position = new
                Vector3(float.Parse(positionXYZ[0]),
                float.Parse(positionXYZ[1]),
                float.Parse(positionXYZ[2]));
            newCube.transform.localScale = new
                Vector3(float.Parse(scaleXYZ[0]),
                float.Parse(scaleXYZ[1]), float.Parse(scaleXYZ[2]));
            cubes.Add(newCube);
        }
        create = false;
    }

    if(update)
    {
        updateMessages = queue.Dequeue();
        for(int i = 0; i < updateMessages.Length; i++){
            int id = updateMessages[i].cubeID;
            string [] rotationXYZ =
                updateMessages[i].rotation.Split('|');
            cubes[id].transform.eulerAngles = new Vector3(
                float.Parse(rotationXYZ[0]),
```

```
                float.Parse(rotationXYZ[1]),
                float.Parse(rotationXYZ[2])
            );
        }
        SendAcknowledge();
        update = false;
    }
}
```

Listato 4.9: Funzioni Start() e Update().

4.3 Strategia basata su protocollo UDP

Nell'ambito del progetto di tesi, è stata effettuata un'esplorazione anche su un'architettura basata su UDP. Come già anticipato questa soluzione è molto simile a quella precedentemente illustrata, il server UDP è stato realizzato in **Java** sempre utilizzando il framework **Vert.x**, mentre il client è stato sviluppato in **C#**. Lato server per realizzare una comunicazione UDP è stata utilizzata la classe `DatagramSocket` [32] al posto della classe `NetServer` utilizzata nell'implementazione TCP e analogamente all'esempio già presentato, il server è stato messo in ascolto sull'indirizzo locale **192.168.40.100** con numero di porta **10000**. Per quanto riguarda il client, esso è stato connesso tramite la classe `DatagramSocket` al server, utilizzando coerentemente l'indirizzo ip e il numero di porta precedentemente illustrati.
Questa soluzione avrebbe portato sicuramente vantaggi dal punto di vista delle performance, essendo UDP un protocollo *connection-less* e strutturalmente meno elaborato di TCP. Sarebbe stato però necessario considerare che utilizzando UDP non vengono garantiti vari aspetti della comunicazione come sicurezza o affidabilità e ciò avrebbe potuto introdurre delle problematiche. Ad esempio sarebbe stato necessario gestire la perdita di pacchetti o l'ordinamento con il quale vengono ricevuti lato client, situando però il caso di studio in un contesto di rete locale si sarebbero potute esaminare queste problematiche con strategie non particolarmente complicate.
Purtroppo però durante l'esplorazione e la realizzazione dell'infrastruttura che utilizza UDP, si sono riscontrate incompatibilità e mancanza di supporto tecnico tra il protocollo e il dispositivo in utilizzo ovvero HoloLens 2. Le tecnologie utilizzate nel progetto di tesi sono ancora in fase di espansione e allo stato attuale non hanno permesso la realizzazione completa di un'infrastruttura che utilizza UDP come protocollo di comunicazione. Nel prossimo capitolo si procede dunque ad esaminare nel dettaglio l'architettura realizzata con il protocollo TCP, verificandone le prestazioni in varie situazioni ritenute significative.

Capitolo 5

Analisi delle performance

In questo capitolo verranno esaminati a livello tecnico i risultati ottenuti durante la realizzazione del progetto di tesi. Come già illustrato, UDP non presenta il supporto tecnico necessario per essere esaminato sotto questo frangente, per cui ci si concentrerà su una solida analisi dell'architettura basata su TCP, in varie situazioni rilevanti. Utilizzando un approccio ingegneristico, per poter avere un riscontro scientifico delle performance dell'architettura progettata è necessario mettere in campo strategie che coinvolgono alcuni concetti matematici e statistici di base. In generale un aspetto che verrà sicuramente considerato, riguarda la latenza che viene introdotta da TCP tra un cambiamento di stato nel Model e la conseguente renderizzazione di esso nella parte di View. Questa caratteristica è sicuramente visibile ad occhio nudo, ma è necessario effettuare delle misurazioni oggettive di questo parametro, permettendo così un'eventuale comparazione tecnica con altre tecnologie.

5.1 Strategia di misurazione e scenari rilevanti

Prima di poter effettivamente procedere nella verifica delle performance dell'architettura è necessario formalizzare alcuni concetti indispensabili che saranno utilizzati per raggiungere lo scopo.
Per semplicità consideriamo la presenza di un solo cubo come entità logica nel Model, che periodicamente viene ruotato lungo l'asse x. Possiamo identificare la rotazione del cubo come un effettivo **cambio di stato** dell'oggetto, formalizzando quindi questo concetto avremo generalmente durante l'esecuzione dell'applicazione, la generazione di più stati dell'oggetto S0, S1, S2, ... , Sn.
Ad ogni nuovo stato generato corrisponde un invio tramite TCP delle informazioni relative ad esso al client e una successiva renderizzazione dei cambiamenti. L'entità logica del model viene ruotata, le informazioni relative alla sua rotazione vengono trasmesse al client e l'ologramma relativo al cubo ruota

graficamente. A questo punto esiste un concetto utile per misurare le prestazioni del protocollo durante questo procedimento?
Un concetto fondamentale in questo caso è il **Round Trip Time (RTT)**, esso rappresenta il tempo di andata e ritorno impiegato da un pacchetto all'interno di un'infrastruttura di rete. Un approccio tecnico consisterebbe nel rilevare il tempo in cui viene inviato un pacchetto, rilevare il tempo in cui viene ricevuto un **ACK** lato server successivamente alla renderizzazione del nuovo stato e applicare la seguente formula:

$$dt = RTT/2 = \frac{ACKTimestamp - sendTimestamp}{2}$$

Il numeratore rappresenta il vero e proprio RTT, come tempo di andata e ritorno del pacchetto, conseguentemente il parametro *RTT/2* rappresenta il tempo di sola andata.
Possiamo quindi pensare a *RTT/2* come uno sfasamento temporale *dt* tra il momento in cui avviene un cambiamento del Model e il momento in cui viene renderizzato nella View.
Può risultare inoltre interessante inviare l'ACK esattamente nel momento di ricezione del pacchetto lato client, in modo da permettere la rilevazione delle latenze riferite prettamente alla comunicazione tramite TCP. A questo punto si noti che ad ogni nuovo stato *Si* corrisponde un particolare *dt*.
Per tutte le misurazioni che verranno effettuate nelle sezioni successive sono stati rilevati gli sfasamenti temporali *dt* di tutti i pacchetti inviati, considerando come tempistiche **un minuto** di esecuzione. Una volta effettuate le misurazioni è possibile effettuare semplici calcoli matematici di valutazione delle performance utilizzando i *dt* generati ad ogni cambio di stato.
Un primo semplice calcolo riguarda la **media aritmetica** degli sfasamenti temporali corrispondenti ai vari stati. Per ogni cambio di stato vengono quindi campionati i tempi e di conseguenza calcolato il parametro *RTT/2* che viene identificato come *dt*, a tutti i *dt* corrispondenti ai vari stati viene applicata la seguente formula che determina lo sfasamento temporale medio:

$$\overline{dt} = \frac{dt0 + dt1 + dt2 + ... + dtn}{n}$$

Un altro concetto fondamentale che permette di determinare in maniera rigorosa le prestazioni del protocollo è la **varianza**. Date tutte le misurazioni effettuate essa permette di determinare lo scostamento dei dati dal valore medio registrato. La varianza viene calcolata utilizzando la seguente formula:

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n}$$

Un ultimo calcolo che può fornire dati interessanti riguarda il tempo impiegato per il **rendering** nelle varie situazioni. Possiamo calcolare la **media aritmetica** delle latenze includendo il rendering nella misurazione ($\overline{dt}_{rendering}$) e successivamente calcolare la **media aritmetica** delle latenze relative alla sola comunicazione TCP ($\overline{dt}_{TCP}$), effettuando la sottrazione tra i due membri si otterrà una stima del tempo medio impiegato nel rendering.

$$\overline{T}_{rendering} = \overline{dt}_{rendering} - \overline{dt}_{TCP}$$

Una volta determinati i parametri secondo i quali verrà valutata l'architettura, ovvero **media aritmetica** e **varianza**, è necessario esaminare quali possono essere degli scenari di misurazione significativi.
Sicuramente il punto di partenza è rappresentato dalla presenza di un solo cubo che come cambiamento di stato considera la propria rotazione lungo un asse cartesiano, successivamente lo scenario è espandibile sotto varie dimensioni. Una prima espansione può riguardare il **numero di ologrammi**, vengono quindi valutate situazioni in cui sono presenti più cubi che cambiano stato (non necessariamente tutti nello stesso modo), e viene quindi testata la capacità dell'architettura di sostenere più entità contemporaneamente.
Un'altra espansione importante, riguarda la **modalità di notifica degli aggiornamenti**. In generale è possibile adottare più strategie, inizialmente si può considerare di inviare periodicamente lo stato dell'intero mondo digitale al client, indipendentemente dal fatto che le entità logiche abbiano subito cambiamenti o meno. Successivamente è possibile pensare ad un'ottimizzazione per la quale vengono inviati solamente i dati riguardanti le entità che hanno effettivamente subito un cambiamento, ed è perciò necessario un aggiornamento della loro parte grafica. Diventa quindi interessante comparare le medie e le varianze calcolate in scenari con uno o più entità e valutare l'efficacia dell'ottimizzazione rispetto alla versione base. Infine ci si potrebbe concentrare sullo sfasamento temporale che si verifica mettendo in campo due client diversi collegati al server.

5.2 Aggiornamento base

In questa categoria di misurazioni, viene considerato un primo scenario di base utilizzando il server TCP esaminato nella sezione 4.2.1. La caratteristica fondamentale che viene messa in evidenza in questa fase riguarda il fatto che il server periodicamente invia un array JSON che contiene gli aggiornamenti riguardanti **tutto il mondo digitale**. In questo caso quindi il client riceve costantemente le indicazioni riguardo alla posizione, la rotazione e la scala di tutti i cubi, indipendentemente dal fatto che abbiano subito dei cambiamenti o

meno. Questa soluzione risulta più semplice dal punto di vista implementativo, e può essere interessante esaminare la variazione delle latenze in comparazione con una versione del server ottimizzata e di conseguenza più complessa. Di seguito si esamina la scalabilità del sistema rispetto al numero di entità presenti, in scenari di complessità crescente.

5.2.1 Scenario 1 : Un cubo

Questo scenario rappresenta la situazione più semplice in cui il sistema possa trovarsi. Viene utilizzata una comunicazione non ottimizzata, ed è presente un solo cubo che ruota con velocità costante.

Figura 5.1: Scenario di base con un solo cubo.

A seguito delle misurazioni delle latenze, vengono riportati i risultati delle medie aritmetiche considerando anche le latenze dovute al rendering dell'ologramma:

$$\overline{dt}_{TCP} = \frac{dt0 + dt1 + dt2 + ... + dtn}{n} = \mathbf{12.102\ ms}$$

$$\overline{dt}_{rendering} = \frac{dt0 + dt1 + dt2 + ... + dtn}{n} = \mathbf{12.955\ ms}$$

Successivamente è possibile determinare le varianze:

$$\sigma^2_{TCP} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n} = \mathbf{84.567\ ms^2}$$

$$\sigma^2_{rendering} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n} = \mathbf{87.065\ ms^2}$$

Notiamo che non è presente una particolare differenza tra la latenza media introdotta dalla sola comunicazione TCP e quella che include il rendering, e anche la varianza subisce un incremento minimo. Di seguito viene calcolato l'effettivo tempo impiegato per la renderizzazione del cubo:

$$\overline{T}_{rendering} = \overline{dt}_{rendering} - \overline{dt}_{TCP} = \mathbf{0.853ms}$$

5.2.2 Scenario 2 : 10 cubi

Espandendo lo scenario dal punto di vista del numero di cubi, è possibile testare la scalabilità del sistema per quanto riguarda le entità presenti al suo interno.

$$\overline{dt}_{TCP} = \frac{dt0 + dt1 + dt2 + ... + dtn}{n} = \mathbf{13.111\ ms}$$

$$\overline{dt}_{rendering} = \frac{dt0 + dt1 + dt2 + ... + dtn}{n} = \mathbf{64.020\ ms}$$

$$\sigma^2_{TCP} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n} = \mathbf{102.640\ ms^2}$$

$$\sigma^2_{rendering} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n} = \mathbf{253.922\ ms^2}$$

Già con il passaggio a 10 cubi possiamo notare un notevole aumento della latenza di trasmissione e della varianza che includono il rendering, bisogna tenere in considerazione che per ogni cubo presente nel sistema viene trasmesso un oggetto JSON che contiene 4 proprietà al suo interno, che vengono decodificate, interpretate e utilizzate per la renderizzazione.

$$\overline{T}_{rendering} = \overline{dt}_{rendering} - \overline{dt}_{TCP} = \mathbf{50.909\ ms}$$

Con l'aumento del numero di entità all'interno del Model, notiamo un incremento conseguente del tempo di renderizzazione, mentre la latenza della comunicazione TCP rimane intorno a 13 millisecondi.

5.2.3 Scenario 3 : 30 cubi

Consideriamo ora il caso pessimo, incrementando il numero di cubi a 30 ed eseguendo calcoli analoghi agli scenari precedenti.

$$\overline{dt}_{TCP} = \frac{dt0 + dt1 + dt2 + ... + dtn}{n} = \mathbf{13.203\ ms}$$

$$\overline{dt}_{rendering} = \frac{dt0 + dt1 + dt2 + ... + dtn}{n} = \mathbf{127.268\ ms}$$

$$\sigma^2_{TCP} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n} = \mathbf{106.902\ ms^2}$$

$$\sigma^2_{rendering} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n} = \mathbf{728.629\ ms^2}$$

Considerando uno scenario in cui sono presenti 30 cubi si iniziano ad ottenere delle latenze non indifferenti per quanto riguarda la trasmissione e la renderizzazione, intorno a 127 ms. In questo contesto si stanno prendendo in considerazione entità molto semplici che non possiedono comportamenti elaborati, nonostante ciò è possibile rendersi conto del fatto che sia ridondante l'invio di informazioni su entità che non hanno subito mutamenti, con la conseguente renderizzazione dell'intero mondo digitale.

$$\overline{T}_{rendering} = \overline{dt}_{rendering} - \overline{dt}_{TCP} = \mathbf{114.065\ ms}$$

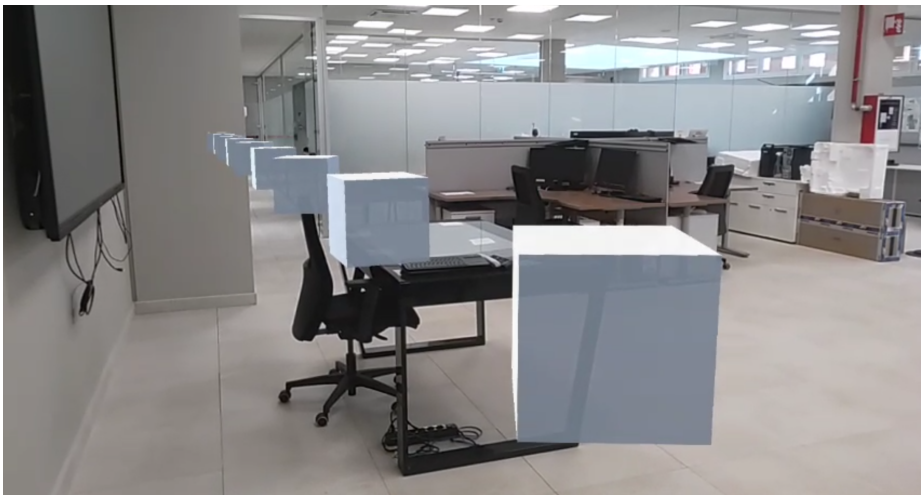

Figura 5.2: Scenario avanzato contenente più cubi.

5.3 Aggiornamento ottimizzato

In questa sezione viene effettuata una semplice ottimizzazione della soluzione proposta nelle fasi precedenti. Come già accennato, si fa riferimento al fatto che inviare periodicamente lo stato di tutto il mondo digitale sia una soluzione più semplice ma allo stesso tempo ridondante. Perciò sono state fatte delle modifiche al server, in modo da poter trasmettere al client i dati relativi ai soli cubi che hanno subito dei cambiamenti.

```
for(int i= 0; i < NUMBER_OF_CUBES; i++) {
    if(!toUpdate[i]) {
        continue;
    }
    JsonObject cubeJson = new JsonObject();
    cubeJson.put("cubeID", cubes.get(i).getId());
    cubeJson.put("position", cubes.get(i).getPosition().toString());
    cubeJson.put("rotation", cubes.get(i).getRotation().toString());
    cubeJson.put("scale", cubes.get(i).getScale().toString());
    packetJson.add(cubeJson);
    toUpdate[i] = false;
}
```

Listato 5.1: Inserimento dei soli cubi necessari

È stata creata una lista di booleani, in cui ogni elemento di essa indica se il cubo corrispondente ha subito una modifica e necessita quindi di un aggiornamento grafico. Perciò tutte le volte che viene modificato un cubo, viene settato a true il corrispondente flag nella lista `toUpdate` e nella fase di composizione del pacchetto JSON (vedi listato 5.1) vengono inseriti in esso solo i cubi che hanno il flag a true.
Con questa metodologia, il client riceve solamente le informazioni necessarie agli aggiornamenti grafici da effettuare, alleggerendo la comunicazione e l'interpretazione dei pacchetti.
Gli scenari seguenti sono stati organizzati in modo che solamente **metà dei cubi** subiscano dei mutamenti a livello di Model, di conseguenza sarà necessario renderizzare unicamente gli ologrammi relativi alle entità che hanno subito dei cambiamenti. Nelle sezioni successive si esaminano i risultati ottenuti con questa ottimizzazione, nei medesimi scenari esaminati nella fase precedente.

5.3.1 Scenario 1 : 10 cubi

Come primo scenario con la comunicazione ottimizzata esaminiamo quello che contiene 10 cubi. Il caso base in cui è presente un solo cubo non risulta rilevante in quanto troppo semplice per poter effettivamente percepire i benefici delle modifiche effettuate. Come già anticipato, vengono trasmesse solamente le informazioni relative alle entità che subiscono mutamenti, quindi in questo caso vengono considerati gli aggiornamenti di **5 cubi**.

$$\overline{dt}_{TCP} = \frac{dt0 + dt1 + dt2 + ... + dtn}{n} = \mathbf{13.008\ ms}$$

$$\overline{dt}_{rendering} = \frac{dt0 + dt1 + dt2 + ... + dtn}{n} = \mathbf{35.096\ ms}$$

$$\sigma^2_{TCP} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n} = \mathbf{66.140\ ms^2}$$

$$\sigma^2_{rendering} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n} = \mathbf{70.564\ ms^2}$$

Esaminando i risultati si nota un notevole miglioramento nelle latenze che comprendono il tempo di rendering, si è passati da una latenza media di **64 ms** ad un ritardo di **35 ms**, anche la varianza (sempre relativa alla renderizzazione) è diminuita notevolmente passando da circa **254 ms**^{2} ad un valore di **66 ms**^{2}. Per quanto riguarda le latenze di TCP, come ci si poteva aspettare, sono rimaste bilanciate attorno ad un valore di 13 millisecondi.

$$\overline{T}_{rendering} = \overline{dt}_{rendering} - \overline{dt}_{TCP} = \mathbf{22.088\ ms}$$

5.3.2 Scenario 2 : 30 cubi

Consideriamo ora lo scenario più complesso contenente 30 cubi per poter valutare le prestazioni dell'ottimizzazione in una situazione più onerosa rispetto alla precedente.

$$\overline{dt}_{TCP} = \frac{dt0 + dt1 + dt2 + ... + dtn}{n} = \mathbf{13.114\ ms}$$

$$\overline{dt}_{rendering} = \frac{dt0 + dt1 + dt2 + ... + dtn}{n} = \mathbf{68.173\ ms}$$

Si noti come le latenze relative a TCP rimangano ancora una volta intorno a 13 millisecondi, anche la varianza relativa ad esso non subisce particolari variazioni.

$$\sigma^2_{TCP} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n} = \mathbf{105.231\ ms^2}$$

$$\sigma^2_{rendering} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n} = \mathbf{110.470\ ms^2}$$

I vantaggi dell'ottimizzazione si percepiscono principalmente per quanto riguarda le tempistiche che comprendono il rendering degli ologrammi, inoltre i vantaggi ottenuti sono più evidenti all'aumento della complessità dello scenario. Il ritardo di trasmissione con rendering passa da **127 ms** a **68 ms** venendo quasi dimezzato mentre la varianza passa da **729 ms**^{2} a un valore di **110 ms**^{2}.

$$\overline{T}_{rendering} = \overline{dt}_{rendering} - \overline{dt}_{TCP} = \mathbf{55.059\ ms}$$

Infine notiamo che, come ci si poteva aspettare, il tempo di rendering della versione ottimizzata è pari a circa **55 ms**, mentre nel medesimo scenario considerando la versione base si aggirava intorno a **114 ms**.
Questo accade perchè come già esplicitato, nella versione ottimizzata vengono inviate le informazioni relative alle sole entità che hanno subito un mutamento, che in questo caso di studio sono esattamente la metà ovvero 15 cubi.

5.4 Più client

Come scenario conclusivo è stato considerato l'obbiettivo di questo caso di studio, ovvero un'esperienza condivisa. Sono stati utilizzati due HoloLens 2 come client per verificare le prestazioni dell'architettura anche nel frangente della condivisione. Sono state apportate delle modifiche al server per permettere, tramite le socket, la connessione di più client e l'invio da parte del server delle informazioni di aggiornamento in broadcast a tutti i client connessi.
In questo esempio non viene considerata un'espansione della complessità dal punto di vista delle entità presenti, ma si è scelto di focalizzare lo studio sulla scalabilità a livello del numero di utenti.
Di conseguenza nello scenario sono presenti **5 cubi** che ruotano lungo un asse cartesiano e sono state rilevate le latenze di trasmissione in maniera analoga agli scenari già presentati.

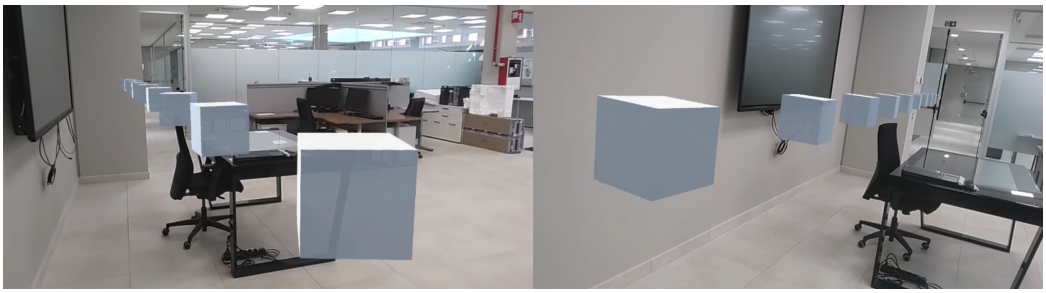

Figura 5.3: Scenario multi client.

Una volta rilevate le latenze percepite dai due client durante l'esecuzione è stata applicata la seguente formula:

$$\overline{dt}_{diff} = \frac{\sum_{i=1}^{n}(|x_{client1} - x_{client2}|)}{n} = \mathbf{21.5\ ms}$$

Entrambi i client ricevono le informazioni di aggiornamento con una determinata latenza. Il dato interessante da considerare in questo scenario è lo **sfasamento temporale** presente tra i due client, esso ci da una valenza oggettiva di quanto effettivamente l'esperienza venga percepita in contemporanea dai due utenti.
Di conseguenza definiamo lo sfasamento temporale come $|x_{client1} - x_{client2}|$ e lo utilizziamo per calcolare il parametro dt_{diff}, che non è altro che la **media aritmetica** degli sfasamenti temporali tra i due client. Oltre alla media aritmetica degli sfasamenti temporali, risulta utile calcolare anche la **varianza** relativa ad essi:

$$\sigma^2 = \frac{\sum_{i=1}^{n}(x_i - \overline{x})^2}{n} = \mathbf{180.5\ ms^2}$$

La soluzione architetturale adottata con ottimizzazione della comunicazione, presenta uno sfasamento temporale medio tra i due client pari a circa **22 ms** che è sicuramente un risultato accettabile.

5.5 Considerazioni finali e possibili estensioni

All'interno di quest'ultimo capitolo, sono stati ottenuti vari dati che è necessario interpretare e su cui è opportuno fare delle considerazioni.
In generale si è notato che le **latenze** introdotte dalla sola comunicazione TCP rimangono **costanti** all'aumentare della complessità dello scenario, questo sicuramente è dovuto al fatto che i pacchetti trasmessi in formato JSON non sono dimensionalmente penalizzanti.
Evolvendo il caso di studio rispetto alla complessità delle entità all'interno del Model, si potrebbe raggiungere la necessità di ottimizzare le dimensioni dei pacchetti eventualmente con delle tecniche di **compressione**.
La fase di rendering è quella che più viene penalizzata dall'aumento del numero di cubi, per cui l'**ottimizzazione della comunicazione** è stata assolutamente efficacie nel miglioramento delle prestazioni del sistema.
Per quanto riguarda lo scenario multi client, sono stati ottenuti risultati più che soddisfacenti con latenze che dimostrano un buon **sincronismo di visualizzazione** degli ologrammi.
Il caso di studio affrontato rappresenta solamente un inizio di sperimentazione nell'ambito delle architetture per sistemi di Mixed Reality condivisi.
In questo elaborato sono state progettate le basi per poter effettuare esplorazioni in questo campo sotto diversi aspetti.
In generale possono essere valutate espansioni rispetto all'infrastruttura di rete, effettuando verifiche del funzionamento in un contesto di rete differente rispetto alla rete locale e sicuramente più aderente alla realtà.
Un'altra estensione interessante può riguardare l'aggiunta di complessità a livello di Model, realizzando un mondo digitale che presenta entità differenti tra loro, che possiedono una logica comportamentale più complessa e richiedono quindi una maggior quantità di dati di aggiornamento.
Si potrebbe inoltre pensare di realizzare un'architettura che utilizza middleware dedicati alla realizzazione di esperienze condivise, come ad esempio Croquet [33], e compararne le prestazioni con l'architettura TCP.
Un'ulteriore valutazione interessante potrebbe essere effettuata aggiungendo la possibilità per i client di modificare gli ologrammi e propagare la modifica al Model, esaminando le performance e le latenze rilevate dagli altri client connessi durante la renderizzazione.
In conclusione, sono tanti gli aspetti in cui questo progetto può subire estensioni interessanti, che si espandono in ambiti differenti e possono essere sviluppate coerentemente con il contesto di interesse.

Conclusioni

Concludendo, in questa tesi è stato esaminato in primo luogo il panorama tecnologico nell'ambito delle esperienze condivise di Mixed Reality.
Analizzando lo stato dell'arte si evince il fatto che tutte le tecnologie presenti al giorno d'oggi nell'ambito delle esperienze condivise, hanno una forte dipendenza da infrastrutture cloud. Ciò aggiunge complessità architetturale e rende ovviamente necessario il collegamento a servizi forniti da entità esterne, come ad esempio Microsoft.
Questo porta a porsi un interrogativo: è possibile realizzare un'architettura senza dipendenze da servizi cloud, che permetta la condivisione di un'esperienza di Mixed Reality? E che prestazioni ha?
Traendo delle conclusioni riguardo al lavoro svolto nel progetto di tesi, sono state applicate le conoscenze acquisite durante i vari studi di questi anni ed è stata realizzata un'architettura client-server in rete locale, utilizzando il pattern di progettazione MVC, una logica di comunicazione basata sul polling e il protocollo TCP. Anche se si tratta di un'architettura molto semplice essa si è rivelata efficacie: tramite vari scenari significativi di complessità crescente, si sono verificate le prestazioni del sistema dal punto di vista della scalabilità a livello di ologrammi e di utenti, ottentendo misurazioni oggettive di latenze medie e varianze riguardanti i ritardi introdotti dal solo protocollo TCP e quelli che considerano anche il tempo di rendering.
Analizzando i dati ottenuti si può concludere che è possibile progettare architetture per sistemi di Mixed Reality condivisi che non utilizzano il cloud e forniscono buone prestazioni sia dal punto di vista delle latenze di comunicazione che per quanto riguarda latenze di rendering e sincronismo tra client differenti. Sarebbe inoltre interessante migliorare la soluzione realizzata sotto vari aspetti, rendendola sempre più aderente ad un contesto applicativo realistico ed eventualmente confrontarla con soluzioni architetturali che adottano strategie differenti.
L'avanzamento tecnologico sarà determinante nel permettere la realizzazione di studi su diverse alternative architetturali, che possono risultare preferibili a strutture dipendenti dal cloud, per motivi di semplicità o prestazioni in diverse situazioni e contesti applicativi.

Ringraziamenti

Innanzitutto vorrei ringraziare il professor Alessandro Ricci e il dottor Samuele Burattini che mi hanno guidato e supportato durante lo sviluppo della tesi, facendomi interessare e appassionare agli studi e alle ricerche di cui si occupano.
Ringrazio inoltre la mia famiglia, che mi ha sopportato anche nei momenti più difficili di questo percorso universitario ed è a loro che dedico questo traguardo.
Un ringraziamento speciale va al mio migliore amico di una vita, per tutte le parole dette in questi anni ma soprattutto per quelle di cui non c'è stato bisogno, grazie.
Un grazie anche ai miei amici di San Mauro, che hanno riempito le mie giornate di spensieratezza e leggerezza.
Infine ringrazio i miei colleghi dell'università con cui ho condiviso il percorso di questi tre anni, augurando a tutti che il futuro riservi le opportunità per cui stiamo sognando.

Bibliografia

[1] Dilian Alejandra Zuniga Gonzalez, Deborah Richards, Ayse Aysin Bilgin. *"Making it Real: A Study of Augmented Virtuality on Presence and Enhanced Benefits of Study Stress Reduction Sessions"*, 2020.

[2] C. Anthes, R. J. García-Hernández, M. Wiedemann and D. Kranzlmüller, "State of the art of virtual reality technology," 2016 IEEE Aerospace Conference, 2016, pp. 1-19, doi: 10.1109/AERO.2016.7500674.

[3] Carmigniani, J., Furht, B., Anisetti, M. et al. Augmented reality technologies, systems and applications. Multimed Tools Appl 51, 341–377 (2011). https://doi.org/10.1007/s11042-010-0660-6

[4] A. Ricci, M. Piunti, L. Tummolini, C. Castelfranchi. *"The MirrorWorld: Preparing for Mixed-Reality Living"* (2015).

[5] Elmahal, D., Ahmad, A., Alomaier, A., Abdlfatah, R., Hussein, D. (2020). Comparative Study between Hologram Technology and Augmented Reality. Journal of Information Technology Management, 12(2), 90-106. doi: 10.22059/jitm.2020.75794

[6] Ong, S., Siddaraju, V.K. (2021). Using Spatial Awareness. In: Beginning Windows Mixed Reality Programming. Apress, Berkeley, CA.

[7] "Spatial coordinate systems." `https://docs.microsoft.com/en-us/windows/mixed-reality/design/coordinate-systems`

[8] Ong, S., Siddaraju, V.K. (2021). Interacting with Holograms. In: Beginning Windows Mixed Reality Programming. Apress, Berkeley, CA.

[9] E. Costanza, A. M. Kunz, and M. Fjeld, "Mixed reality: A survey," in Human Machine Interaction, 2009.

[10] Ungureanu, Dorin and Bogo, Federica and Galliani, Silvano and Sama, Pooja and Duan, Xin and Meekhof, Casey and Stühmer, Jan and Cashman, Thomas J. and Tekin, Bugra and Schönberger, Johannes L. and Olszta, Pawel and Pollefeys, Marc, *"HoloLens 2 Research Mode as a Tool for Computer Vision Research"*, 2020.

[11] "Hololens 2 Tecnical Specifications" `https://www.microsoft.com/en-us/hololens/hardware`

[12] "What is Mixed Reality Toolkit 2?" `https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05`

[13] "MRTK packages" `https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/packages/mrtk-packages?view=mrtkunity-2021-05`

[14] "Unity Real-Time Development Platform" `https://unity.com/`

[15] "MRTK2 profile configuration guide" `https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/configuration/mixed-reality-configuration-guide?view=mrtkunity-2022-05`

[16] Ryan Anthony J. de Belen, Huyen Nguyen, Daniel Filonik, Dennis Del Favero, Tomasz Bednarz. A systematic review of the current state of collaborative mixed reality technologies: 2013–2018[J]. AIMS Electronics and Electrical Engineering, 2019, 3(2): 181-223. doi: 10.3934/ElectrEng.2019.2.181

[17] Mark Billinghurst, Hirokazu Kato. Collaborative Mixed Reality. Human Interface Technology Laboratory, University of Washington, 2007.

[18] "Shared experiences in mixed reality" `https://docs.microsoft.com/en-us/windows/mixed-reality/design/shared-experiences-in-mixed-reality`

[19] "Developing for HoloLens: Remote Collaboration with Multiple Avatars" `https://vimeo.com/160704056`

[20] Sergio Orts-Escolano, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip L. Davidson, Sameh Khamis, Mingsong Dou, Vladimir Tankovich. 2016. *Holoportation: Virtual 3D Teleportation in Real-time. In Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (UIST '16). Association for Computing Machinery, New York, NY, USA, 741–754. https://doi.org/10.1145/2984511.2984517

[21] *"Holoportation: virtual 3D teleportation in real-time (Microsoft Research)"* `https://www.youtube.com/watch?v=7d59O6cfaM0`

[22] *"Introducing Microsoft Mesh"* `https://www.youtube.com/watch?v=Jd2GK0qDtRg`

[23] *"Microsoft Mesh Overview"* `https://docs.microsoft.com/en-us/mesh/overview`

[24] *"Microsoft Mesh a technical overview"* `https://techcommunity.microsoft.com/t5/mixed-reality-blog/microsoft-mesh-a-technical-overview/ba-p/2176004`

[25] *"JoinXR"* `https://fracturereality.io/projects/joinxr/`

[26] *"About Azure Remote Rendering"* `https://docs.microsoft.com/en-us/azure/remote-rendering/overview/about`

[27] Freeman, A., Sanderson, S. (2011). The MVC Pattern. In: Pro ASP.NET MVC 3 Framework. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4302-3405-0_4

[28] G. Xylomenos and G. C. Polyzos, "TCP and UDP performance over a wireless LAN," IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320), 1999, pp. 439-446 vol.2, doi: 10.1109/INFCOM.1999.751376.

[29] *"Eclipse Vert.x Reactive applications on the JVM"* `https://vertx.io/`

[30] *"Class AbstractVerticle documentation"* `https://vertx.io/docs/apidocs/io/vertx/core/AbstractVerticle.html`

[31] Felipe Pezoa, Juan L. Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. 2016. Foundations of JSON Schema. In Proceedings of the 25th International Conference on World Wide Web (WWW '16). International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 263–273. https://doi.org/10.1145/2872427.2883029

[32] *"Interface DatagramSocket documentation"* `https://vertx.io/docs/apidocs/io/vertx/core/datagram/DatagramSocket.html`

[33] *"Croquet OS - the Operating System for the Metaverse"* `https://croquet.io/`