ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

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

REALIZZAZIONE DI UN FRAMEWORK PER L'INTEGRAZIONE DELLE PIATTAFORME CROQUET E HOLOLENS A SUPPORTO DELLO SVILUPPO DI APPLICAZIONI DI MIXED REALITY CONDIVISA

Elaborato in
SISTEMI EMBEDDED E INTERNET-OF-THINGS

Relatore
Prof. ALESSANDRO RICCI

Corelatore
Dott. SAMUELE BURATTINI

Presentata da
LUCA TONELLI

Anno Accademico 2021 – 2022

Ai miei nonni,
per tutto l'affetto che mi avete sempre dato

Indice

Introduzione

Il continuo svilupparsi della tecnologia negli anni ha portato a raggiungere traguardi impensabili, inutile dire che pochi anni fa non era nemmeno immaginabile la stragrande quantità e complessità di tecnologie che si hanno alla mano al giorno d'oggi.

Tale sviluppo ha portato a una totale rivoluzione di ogni aspetto della vita quotidiana, integrando nella vita di tutti i giorni dispositivi e tecnologie impensabili, ciò ha comportato lo sviluppo, l'adattamento e la rivoluzione degli stessi ambienti che fanno parte della quotidianità. Una delle tecnologie che più colpisce anche al giorno d'oggi sembrando pura fantascienza è la possibilità di integrare veri e propri elementi virtuali nel mondo che ci circonda, permettendo quindi di cambiare totalmente la visione di realtà, consentendo l'interazione con oggetti non visibili nel mondo reale ma solo attraverso un visore o un dispositivo dedicato, andando a realizzare dei veri e propri ologrammi.

Questa tecnologia è chiamata Mixed Reality, ovvero la mescolanza degli oggetti reali all'insieme di oggetti virtuali, integrando alla vita di tutti i giorni veri e propri ologrammi con i quali interagire, ma perché limitare questa tecnologia a un semplice utilizzo nella propria quotidianità ?

Un aspetto veramente interessante che comporta un notevole sviluppo dal punto di vista rivoluzionario e tecnologico è la possibilità di utilizzare tale tecnologia in modo collaborativo, portando quindi più persone a collaborare attraverso uno spazio virtuale, permettendo di oltrepassare i limiti che comporta la realtà. Tale possibilità è quindi resa possibile attraverso applicazioni di Mixed Reality condivise, che permettono il collegamento di più utenti all'applicazione per una gestione collaborativa degli ologrammi sia in locale che da remoto.

Nella tesi corrente verranno quindi esplorati gli aspetti che compongono dal punto di vista teorico e pratico un'applicazione di Mixed Reality condivisa e la relativa implementazione tramite la piattaforma Croquet.

La tesi è disposta nel seguente modo:

- **Capitolo I**: Introduzione degli aspetti teorici relativi alla Mixed Reality, alla piattaforma Croquet e alla Mixed Reality condivisa.

- **Capitolo II**: Introduzione dell'idea di progetto, analizzando nello specifico i concetti di progettazione e le scelte prese.
- **Capitolo III**: Esposizione dell'implementazione relativa al Bridge di comunicazione realizzato.
- **Capitolo IV**: Esposizione dell'implementazione relativa al Client di Mixed Reality.
- **Capitolo V**: Dimostrazione del progetto realizzato tramite l'esposizione di due esempi.
- **Capitolo VI**: Analisi dei risultati ottenuti definendo le problematiche e i lavori futuri.

Capitolo 1

Definizione dei concetti essenziali

In questo primo capitolo si vogliono introdurre i concetti base necessari per focalizzarsi sul caso di studio. Verrà introdotto in primo luogo il concetto di Mixed Reality, andando a definire i punti fondamentali della realtà mista, la strumentazione necessaria e verrà introdotto il concetto base di questo caso di studio, la Mixed Reality condivisa.

Successivamente verrà introdotta la seconda parte fondamentale di questo studio, ovvero la piattaforma Croquet OS, e verranno esposti i punti base della piattaforma, sviluppando in fase finale una focalizzazione sui Microversi.

1.1 Mixed Reality

La Mixed Reality (MR), o realtà mista, è una tecnologia che fonde il mondo fisico e quello digitale, realizzando uno scenario in cui oggetti reali e virtuali coesistono e interagiscono in tempo reale, permettendo quindi di visualizzare oggetti virtuali nel mondo reale attraverso appositi dispositivi tra cui visori, occhiali intelligenti e dispositivi mobili, e garantendo all'utente un' interazione con essi come se fossero oggetti fisici e reali.

L'utilizzo della Mixed Reality si sta diffondendo in molti settori, soprattutto, negli ambiti di istruzione, formazione, medicina e progettazione industriale, permettendo a chi la utilizza di entrare in un ambiente immersivo e dedicato. Parlando di realtà mista è necessario definire i due pilastri che ne stanno alla base: la realtà aumentata e la realtà virtuale.

1.1.1 Augmented Reality

Con Augmented Reality (AR), o realtà aumentata, si intende una tecnologia che consente d'integrare elementi virtuali visivi e sensoriali all'ambiente reale tramite una tecnologia olografica, attraverso l'utilizzo di un dispositivo come uno smartphone, un tablet o un visore AR. Più nello specifico, consente, come si evince dal nome, di "aumentare" la realtà concepita sensorialmente e intellettualmente dall'utente, aggiungendo delle informazioni ad essa, offrendo un' interazione in tempo reale con il mondo fisico, integrato con quello digitale, e un'accurata identificazione degli oggetti virtuali e reali.
Ad esempio, guardando un edificio storico, si potrebbe utilizzare la realtà aumentata per visualizzare informazioni sullo schermo del proprio smartphone o, se si utilizza un visore, per generale direttamente davanti ai nostri occhi sotto forma di ologramma [19].

1.1.2 Virtual Reality

Con Virtual Reality (VR), o realtà virtuale, si intende la simulazione di una realtà effettiva, distaccata da quella reale e realizzata da un ambiente digitale, in cui l'utente, attraverso appositi visori dedicati che coprono occhi e solitamente anche le orecchie, viene immerso. Naturalmente, la tipologia e la qualità del visore (che può essere sostituito anche da uno smarthphone opportunamente accessoriato) e l'utilizzo di accessori specifici, definiscono il grado d'immersione nel mondo virtuale. L'utente può perciò passare da poter solamente visionare il mondo virtuale, a poter interagire con gli oggetti in esso attraverso appositi joystick o guanti in grado di catturare e trasmettere i movimenti e le gesture delle mani, ad avere una completa immersione attraverso apposite piattaforme sempre più complesse, in grado di garantire un tracking a 360 gradi del corpo riportando ogni movimento effettuato, compreso lo spostarsi, all'interno del mondo generato [23].

1.1.3 Continuum di virtualità

Come precedentemente esposto, alla base della **Mixed Reality** possiamo identificare due concetti base: la Augmented Reality e la Virtual Reality. La realtà mista fonde questi due concetti base che si trovano ai poli dello *Spettro di realtà mista* 1.1, permettendo quindi il posizionamento di oggetti virtuali (Ologrammi) nel mondo fisico come se ne facessero parte, e la collaborazione con altri utenti grazie alla rappresentazione nel mondo reale sotto forma di avatar [20].

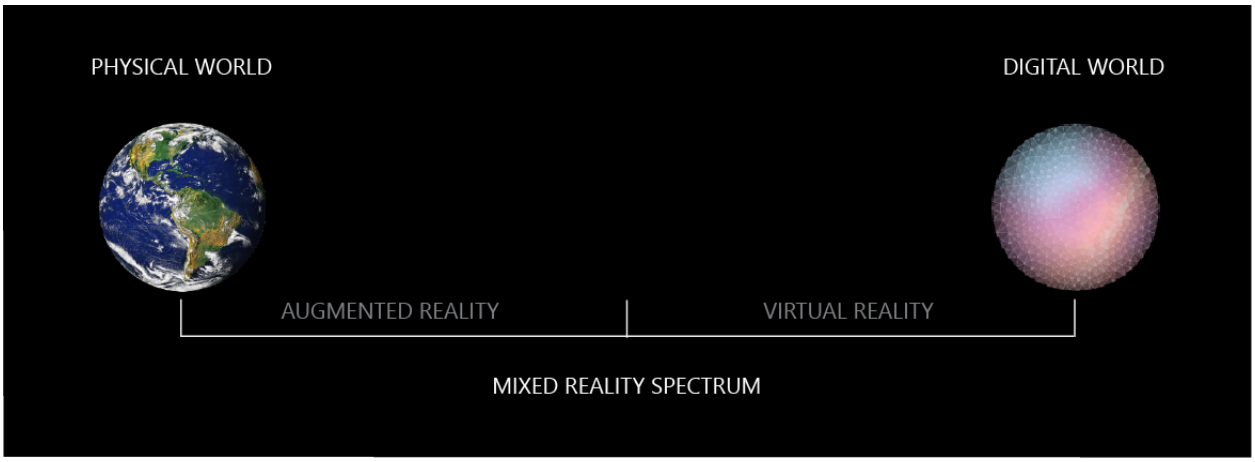

Figura 1.1: Spettro di realtà mista [20]

1.1.4 Ologramma

Nell'ambito della Mixed Reality, per ologramma si intende un' estensione delle immagini olografiche, definite precedentemente, nella realtà aumentata. Mentre prima si parlava d'immagini o modelli olografici visibili attraverso monitor e visori che permettevano di aggiungere informazioni all'ambiente circostante, nella realtà mista questo concetto è stato espanso con la realizzazione di un oggetto tridimensionale composto da luce e suono che può possedere le stesse caratteristiche di un oggetto del mondo reale. Esso può rispondere al tocco, al movimento degli occhi, ai comandi vocali e alle collisioni con gli oggetti reali del mondo che circonda l'utente, permettendogli d'interagire con esso, rendendolo parte integrante del mondo reale.

Il posizionamento di ologrammi nell'ambiente avviene in modo semplice e intuitivo, definendo per ogni ologramma le tre coordinate spaziali X, Y, Z, relative alla posizione iniziale del dispositivo. Nello specifico, la posizione ottimale per l'utilizzo e la visione di ologrammi è compresa tra gli 1,25 metri e i 5 metri di distanza dal visore [2].

1.1.5 Coordinate

Come accennato in precedenza, e come è semplice intuire, il sistema di riferimento di coordinate per i modelli 3D che si tiene in considerazione è quello cartesiano. Vengono definiti quindi tre assi perpendicolari X, Y, Z, associando a ogni ologramma il valore corrispondente dei tre assi XYZ.

Il sistema di coordinate spaziale preso in considerazione esprime i propri valori in metri del mondo fisico, cioè presi in considerazione due oggetti che differenziano di 2 sul valore della coordinata spaziale X, in fase di rendering

essi verranno rappresentati a due metri di distanza l'uno dall'altro sullo stesso asse X.

Esistono in generale due sistemi di coordinate cartesiane: quello *destrorso* e quello *sinistrorso*. In entrambi i sistemi l'unica variazione è la direzione dell'asse Z, che punterà distante da noi nel sistema sinistrorso e verso di noi nel sistema destrorso, mentre l'asse X in entrambi punta verso destra e l'asse Y verso l'alto [11].

1.1.6 Rendering

Per quanto riguarda il rendering olografico la parte fondamentale è sapere con che dispositivo si ha a che fare. Esistono diverse tipologie di dispositivi per la visione di realtà mista, ognuno con caratteristiche differenti. Verranno illustrati i principi del rendering olografico in modo generico, ma si deve tenere in considerazione che lo strumento utilizzato per questo specifico caso di studio è Hololens 2, trattato in seguito.

Esistono differenti tipologie di visualizzazione di ologrammi, alla base delle quali sta la tipologia di display del visore:

- **Display visualizzati:** sono i display montati su dispositivi Hololens. Di base essi aggiungono luce al mondo, rendendo trasparenti i pixel neri, mentre i pixel bianchi sono resi traslucenti perché la luce dai display si mescola con quella del mondo reale, rendendo un pixel più luminoso più opaco.

- **Display opachi:** sono i display montati sui visori VR immersivi. Essi bloccano la visione del mondo rappresentando i pixel con il proprio relativo colore, mentre quelli neri appaiono tutti neri in tinta unita.

Un visore, per renderizzare un' applicazione di realtà mista, deve tenere traccia costantemente della posizione e dell'orientamento della testa. In fase di esecuzione una volta renderizzato il frame corrente, viene preparato il successivo stimando la posizione dell'utente nel futuro nel momento in cui il frame verrà visualizzato, calcolando quindi le proiezioni e le visualizzazioni degli oggetti di tale frame. Per avere una maggiore precisione nel calcolo di frame futuri il sistema misura in modo costante la latenza end-to-end delle pipeline di rendering [15], e possono essere utilizzate dal visore anche applicazioni avanzate per il calcolo e la stima dei frame, le quali hanno la possibilità di effettuare un override sui parametri calcolati dal sistema.

1.1.7 Mapping spaziale

Il mapping spaziale è l'ultimo argomento degno di nota da introdurre per avere una comprensione totale della potenzialità di un'applicazione di Mixed Reality visionata tramite Hololens.

Attraverso il mapping spaziale si è in grado di fornire una rappresentazione dettagliata dell'ambiente reale attorno al visore, permettendo l'unione del mondo virtuale con quello reale. Si consentono quindi il posizionamento di oggetti virtuali su superfici reali e l'ancoraggio di ologrammi a oggetti del mondo fisico, evitando che tutti gli ologrammi seguino l'utente, ma vengano ad esempio occultati da oggetti reali, dando l'illusione che facciano veramente parte del mondo fisico.

Il mapping spaziale è reso possibile grazie a due oggetti fondamentali, che sono *Spatial Surface Observer* e *Spatial Surface*. L'applicazione definisce all'observer delle aree di delimitazione nelle quali si vogliono ricevere dati relativi al mapping spaziale. Se ci si trova nel range indicato all'observer, l'applicazione inizierà a fornire set di superfici spaziali che possono essere di due tipologie differenti:

- **Stazionari:** definiti da una posizione fissa basata sul mondo reale
- **Collegati a Hololens:** elementi che si spostano assieme al visore

Ogni set di superfici spaziali descrive quindi una superfice reale rappresentata sotto forma di Mesh di triangoli. Possiamo quindi iniziare a parlare di una vera e propria consapevolezza spaziale [11].

1.1.8 Dispositivi per la Mixed Reality

Sono presenti differenti dispositivi per accedere alla realtà mista, ogni tipologia permette di accedere e interagire a seconda delle proprie specifiche potenzialità, verranno elencate alcune tipologie in seguito:

- **Head-Mounted Display (HMD)**: Si tratta di occhiali intelligenti che permettono di visualizzare oggetti virtuali sovrapposti al mondo reale. Gli HMD includono dispositivi come HoloLens di Microsoft, Magic Leap One e Meta 2.
- **Visori per la realtà virtuale (VR)**: I visori VR sono dispositivi indossabili che creano un ambiente virtuale totalmente immersivo. Alcuni esempi di visori VR includono Oculus Rift, HTC Vive e PlayStation VR.
- **Proiettori**: Questi dispositivi proiettano immagini virtuali direttamente sulle mani o su un'area di lavoro, senza la necessità di indossare alcun tipo

di dispositivo. Ad esempio, il Leap Motion Controller è un dispositivo che utilizza i sensori per tracciare i movimenti delle mani e permette di interagire con oggetti virtuali [16].

- **Dispositivi mobili**: I dispositivi mobili, come gli smartphone o i tablet, possono essere utilizzati per la Mixed Reality grazie all'utilizzo di app specifiche che permettono di visualizzare oggetti virtuali sovrapposti alla realtà.

1.2 Mixed Reality Codivisa

Descritto il concetto di Mixed Reality è necessario introdurre l'aspetto di collaborazione in questo specifico contesto, essendo questo effettivamente l'obiettivo che si vuole raggiungere [6].

Verranno elencate le problematiche e attenzioni alle quali si va in contro realizzando un applicativo di Mixed Reality condivisa.

1.2.1 Condivisione

Condividere e rendere collaborativa un applicazione di Mixed Reality è sicuramente un aspetto interessante e complesso. Si possono prima di tutto identificare tre principali tipologie di condivisione che vanno a definire il tipo di scenario. La collaborazione può essere effettuata in modalità **Presentazione**, **Collaborazione** e **Guida**. La differenza tra le tre tipologie è abbastanza evidente. Condividendo in modalità Presentazione possiamo identificare una figura di Master, che condivide la propria esperienza, e uno o più Slave in ascolto, che parteciperanno passivamente. Passando alla condivisione di Collaborazione, possiamo, come si intuisce dal nome, identificare non più un Master attivo e degli Slave passivi, ma una rete in cui tutti gli utenti sono Master attivi e possono interagire con l'ambiente di Mixed Reality. Infine per quanto riguarda la condivisione di tipo Guida, si può identificare in un aspetto di collaborazione ma con soli due utenti.

1.2.2 Locazione

Un aspetto fondamentale da tenere in considerazione è la posizione geografica degli utenti che desiderano partecipare alla condivisione. Potranno essere presenti degli utenti che si trovano nello stesso luogo, mentre altri che avranno necessità di collegarsi da remoto.

Nel caso in cui tutti gli utenti si trovino nello stesso spazio fisico la condivisione avviene in **colocated**, mentre trovandosi nella situazione in cui ogni

utente si trovi in uno spazio fisico differente, si avrà una **remote experience**. Naturalmente viene gestito anche il caso in cui il gruppo sia suddiviso in utenti presenti all'interno dello stesso spazio fisico e non.

1.2.3 Dimensione del gruppo

Prima d'introdurre i concetti base relativi alla condivisione è necessario parlare della possibile dimensione del gruppo. Naturalmente, in un contesto di condivisione uno a uno le problematiche risultano limitate, ma passando a gruppi di utenti si può giungere a dei limiti dovuti alla mole di dati da condividere con ogni utente, e anche alla rappresentazione di tutti gli avatar all'interno di uno spazio chiuso. Sono stati identificati i criteri base per le dimensioni di un gruppo, e parleremo di gruppi **piccoli** nel caso in cui il numero dei partecipanti non superi le 7, persone mentre di gruppi **grandi** in caso contrario.

1.2.4 Sincronia o Asincronia

In un contesto di condivisone potremmo dover interagire con differenti oggetti presenti nella scena. Solitamente l'interazione avviene in modo totalmente **sincrono**, ma introducendo il contesto in una prospettiva più ampia, potremmo pensare di dover interagire con oggetti virtuali generati e lasciati da altri utenti, magari nel passato. Basti pensare ad esempio alla possibilità di lasciare nel mondo virtuale una nota; integrando questa possibilità si entra in un ambiente prettamente **asincrono**.

Possiamo perciò parlare di esperienze sincrone nel caso in cui due o più utenti condividano un'esperienza in tempo reale, e di esperienze asincrone nel caso in cui l'esperienza venga condivisa in due momenti temporali differenti. Ciò non nega la possibilità di realizzare esperienze miste.

1.2.5 Ambiente fisico

In un contesto di Mixed Reality, una parte fondamentale è l'ambiente che circonda l'utente o, in questo specifico caso, gli utenti.

Considerando a due possibili utenti che si collegano da due posizioni geograficamente differenti, è prettamente impossibile che entrambi si trovino in stanze totalmente identiche, a meno che non vengano realizzate appositamente per tale scopo; possono però trovarsi in due ambienti **simili**.

Possiamo quindi parlare di:

- **Ambienti simili:** se i due ambienti fisici presentano arredi, luce e rumori ambientali simili, comprese le dimensioni della stanza.

- **Ambienti dissimili:** se gli ambienti in cui si trovano gli utenti non hanno caratteristiche similari e sono quindi totalmente differenti.

La similarità degli ambienti è un aspetto non di poco conto. Possono essere presenti alcune limitazioni date dall' applicazione che si intende condividere, sia dal punto di vista dell'esperienza stessa, per cui si potrebbe ad esempio necessitare di una superficie piana come un tavolo per la corretta esecuzione dell' esperienza, sia dal punto di vista della scala degli oggetti.

1.2.6 Tipologia di dispositivo

Portando il contesto di condivisione a una possibilità di accesso tramite dispositivi che non siano necessariamente visori MR, è necessario porsi questa problematica.

Nella realizzazione di una applicazione è necessario tenere conto delle limitazioni che possono avere i vari dispositivi, naturalmente nel caso in cui ci si trovi in un contesto di dispositivi 2D e 3D che necessitano una collaborazione.

1.2.7 Applicazioni

Di seguito verranno descritti due applicativi di Mixed Reality condivisa.

MESH

Microsoft Mesh è una piattaforma di realtà mista sviluppata da Microsoft, che consente agli utenti di collaborare e interagire in un ambiente virtuale condiviso in tempo reale, indipendentemente dalla loro posizione geografica.

Ogni utente che partecipa alla sessione condivisa viene rappresentato da un avatar tridimensionale. L'interazione tra utenti è resa realistica attraverso l'audio spaziale e la prossimità, consentendo a ognuno di visualizzare il resto del gruppo. Ogni utente ha la possibilità, oltre a poter interagire con gli altri e con gli oggetti condivisi, di realizzare e visualizzare note in uno spazio tridimensionale condiviso [9].

Sono stati definiti alcuni scenari base che possono essere realizzati tramite Mesh [14]:

Collaborazione virtuale : Viene creato un contesto di collaborazione integrato a Microsoft 365 per la sincronizzazione di calendari, contenuti e flussi di lavoro.

Revisioni di progettazione con riconoscimento spaziale : Viene consentito il collegamento da qualsiasi dispositivo, in modo da permettere di

annotare in tempo reale i modelli 3D, il contenuto viene reso persistente e accessibile a tutto il team.

Aiutare gli altri in remoto : Viene consentito un collegamento da remoto ad esempio a scopo di consulenza, in grado quindi di condividere informazioni in modo rapido e sovrapporsi ai dati contestuali.

Eseguire il training e imparare insieme : Vengono realizzate sessioni di insegnamento, in cui viene condiviso il materiale olografico, ogni sessione permette, in un contesto di apprendimento, di trovarsi nella stessa stanza con insegnanti permettendo la visualizzazione dello stesso set di oggetti da più prospettive e utilizzando l' oloportazione, evitando quindi i costi derivanti dai viaggi e dalla logistica.

Ospitare incontri virtuali : Permette di gestire incontri virtuali in un ambiente di Mixed Reality.

Holoportation

Holoportation è una tecnologia che consente di proiettare un'immagine 3D di una persona in un'altra posizione, in modo che sembri che tale persona sia fisicamente presente in quel luogo. La tecnologia utilizza la combinazione di telecamere e sensori di profondità per catturare l'immagine tridimensionale del soggetto e trasmetterla in tempo reale attraverso la rete. Questo consente a una persona di interagire con gli altri in tempo reale come se fosse presente fisicamente nello stesso spazio, anche se si trova in un luogo diverso [8].

Il concetto base è simile a quello di Mesh, cioè la realizzazione di un' esperienza di realtà mista condivisa che permetta però di rendere il contesto il più realistico possibile, venendo non più rappresentati da avatar virtuali 3D ma rappresentati interamente.

1.3 Croquet

Croquet è una piattaforma per creare applicazioni web real-time, le quali consentono a più utenti di collaborare senza la necessità di scrivere codice lato server o di gestire server. La piattaforma è disponibile come libreria JavaScript, che sincronizza le app Croquet attraverso un sistema globale di server reflector che garantiscono un'interattività real-time multi piattaforma tra i vari utenti, questo insieme di caratteristiche rende Corquet una piattaforma ideale per la gestione di un'applicazione di Mixed Reality condivisa.

Il codice client scritto dall' utente viene eseguito all'interno di un virtual machine condivisa (VM). Ogni utente avrà quindi la propria VM in esecuzione

che garantisce l'accesso alla sessione. Per far sì che le varie macchine virtuali si colleghino tra loro, è necessario che siano identiche, garantendo quindi che ogni client abbia lo stesso identico codice (bit a bit) in esecuzione. Le virtual machine andranno poi a comunicare tra loro inviando e ricevendo eventi in modo da eseguire tutte lo stesso frammento di codice nello stesso momento [3].

Questo aspetto di sincronia è dato dalla vasta distribuzione di server Reflector, come accennato, e dalla garanzia di un tempo unico per tutti i server, questo aspetto verrà approfondito in seguito.

1.3.1 Croquet Reflector Network

Il Croquet Reflector Network (CRN) è un sistema di comunicazione peer-to-peer sviluppato in modo da eliminare il sistema di server dedicati. Si tratta infatti di una rete globalmente distribuita di server cloud che comunicano tra loro a intervalli costanti e controllati da un tempo unico. Il sistema si basa su un protocollo di comunicazione che utilizza una combinazione di tecnologie peer-to-peer e client-server per garantire la massima efficienza e affidabilità. Il Reflector Network supporta anche funzionalità di sicurezza avanzate, come la crittografia end-to-end e la gestione degli accessi basata sui ruoli. Essendo distribuiti e dovendo solamente compiere il ruolo di smistamento dei dati, il CRN riesce a garantire una bassa latenza a tutti gli utenti.

Ogni Reflector, oltre a smistare dati e consentire la connessione alle sessioni geograficamente vicine, tiene conto del tempo relativo della simulazione e invia eventi per aggiornare tutti i vari peer al passare del tempo in modo che anch'essi possano continuare l'esecuzione della propria macchina virtuale contenente la stessa sessione [4].

1.3.2 Tempo

Ogni sessione di croquet non ha il concetto di tempo reale. L'unica espressione di tempo che viene utilizzata è il tempo della simulazione, gestito, come descritto, in precedenza dal reflector e inizializzato in fase di lancio della sessione.

Ogni evento ricevuto dal reflector viene salvato con una marcatura temporale, in modo da poter aggiornare all'ultimo valore ricevuto il tempo della simulazione. Nel caso in cui non vengano intercettati eventi, i reflector generano stream di eventi 20 volte al secondo chiamati **heartbeat ticks**, con il solo scopo di aggiornare il tempo di ogni macchina virtuale collegata al Reflector, e quindi far procedere le sessioni con le azioni schedulate. Questa frequenza può essere cambiata in fase di lancio.

1.3.3 Snapshot

All'interno di ogni reflector è presente un sistema di salvataggio dati e dello stato di ogni sessione, chiamato snapshot. Il reflector serializza periodicamente i dati relativi a ogni sessione in una sorta d'istantanea e li salva su cloud. Questo meccanismo, oltre a consentire un salvataggio di stato in caso di disconnessioni o ricaricamenti della pagina, permette a ogni nuovo client di ottenere e aggiornarsi allo stato corrente della sessione (se già in esecuzione). Il ripristino della sessione avviene assieme alla ricostruzione degli eventi intercettati e salvati, in modo da poterne ricostruire alla perfezione lo stato.

1.3.4 Struttura

Una semplice applicazione per Croquet è composta da due componenti fondamentali, la **View** e il **Model**.

Come definito in precedenza, non è necessario scrivere alcun codice relativo alla parte server. L'intero sistema viene quindi gestito in modo totalmente automatico definendo le due classi che estenderanno Model per inserire il codice relativo alla sessione e simulazione, e View per la gestione degli input e output.

Per garantire l'esecuzione della sessione è necessario un terzo componente, la **API Key**, che garantisce l'accesso al sistema di reflector.

Model

La classe Model rappresenta il cuore dell'applicazione. Essa contiene i dati principali, si occupa della sincronizzazione di essi e gestisce gli eventi ricevuti dalle View.

Il Model è implementato in JavaScript e organizzato in modo gerarchico. É possibile scomporre il modello di un'applicazione in più sotto-modelli, ognuno dedicato a un preciso scopo. Il modello principale è unico per ogni applicazione e viene creato all'avvio del sistema, integrando all'interno di esso i vari sotto-modelli generati.

Come si può ben intuire questa classe è quindi la parte che deve essere replicata identica per ogni sessione, garantendo così il collegamento tra i vari utenti. Quando uno stato del modello viene modificato da un utente, la modifica viene inviata a tutti gli altri utenti in tempo reale, garantendo che tutti i partecipanti abbiano la stessa vista dello stato dell'applicazione.

Nello specifico, questa classe intercetta gli eventi pubblicati dalle View e ricevuti tramite il sistema di reflector, aggiorna lo stato attuale dell'applicazione, e può a sua volta generare eventi dedicati alle View per notificare la modifica dei dati. I dati contenuti al suo interno vengono serializzati e utilizzati per la realizzazione di Snapshot.

Prima dell'effettiva esecuzione dei Model all'interno della macchina virtuale è necessario eseguire la loro registrazione, così da specificare al serializzatore la struttura dei dati da serializzare per la realizzazione d'istantanee e la loro ricomposizione [12].

View

La classe View rappresenta quella che si può definire una vera e propria vista della classe Model. Non necessariamente si parla di componenti grafici che rappresentino lo stato del Model, ma di una classe che alla base comunichi e intercetti i cambiamenti del modello.

La classe View è implementata in JavaScript, e può anch'essa presentare un'architettura gerarchica.

Questa classe, perciò, si occupa principalmente di ricevere gli eventi pubblicati dalle altre istanze di sessione e comunicare l'evento al proprio Model generando anch essa un evento, per questioni di sincronizzazione non è possibile alla View scrivere direttamente i dati contenuti nel proprio modello ma solo accedervi in lettura. Viceversa, nel caso in cui la view riceva un evento da parte del proprio Model, che notifichi ad esempio la modifica di alcuni dati, è incaricata di replicare tale evento nel sistema di Reflector, andando ad aggiornare le altre sessioni.

Per quanto riguarda invece l'aspetto grafico di una View, Croquet specifica che, vista l'implementazione generica della classe, che permette di gestire gli eventi di publish e subscribe, non viene definito un Framework specifico per la realizzazione di un'interfaccia utente [18].

API Key

La API Key è il terzo componente fondamentale di un'applicazione Croquet, poiché permette l'accesso al sistema di reflector.

Ogni utente che voglia realizzare un'applicazione deve prima di tutto iscriversi al sito ufficiale di Croquet e richiedere la generazione delle proprie chiavi. In fase di lancio della sessione, inserendo la chiave dedicata al contesto dell'applicazione, si ha accesso al sistema di reflector.

Sessione

Descritte le parti fondamentali che compongono un'applicazione Croquet, possiamo introdurre il concetto di sessione, ovvero la classe Session, che permette il lancio effettivo dell'applicativo.

Il lancio di una sessione avviene attraverso il metodo statico **join**. In fase di lancio di una sessione devono essere specificati alcuni parametri base, tra cui:

- **apiKey**: la chiave generata all'utente che permette l'accesso alla rete di reflector.
- **appId**: una stringa unica identificativa per la sessione.
- **name**: il nome della sessione alla quale si vuole accedere.
- **password**: la password di accesso alla sessione utilizzata per la crittografia degli eventi e snapshot.
- **model**: il model da registrare e utilizzare in questa sessione.

Sono presenti naturalmente altri parametri non essenziali che non verranno elencati, fra cui il parametro view.

Come si può notare, sono presenti due parametri, name e password, che identificano una sessione. Questo perché Croquet permette la creazione di "stanze" dedicate, ogni Sessione può essere divisa in istanze, consentendo la divisione degli utenti in gruppi. In fase di esecuzione viene generato un identificativo per ogni sessione, dato dal nome e un HASH calcolato tramite tutte le classi del Model, che verrà quindi utilizzato per identificare e riconoscere le sessioni compatibili per lo scambio di eventi.

1.3.5 Eventi

Una volta definita la struttura generale di Croquet è necessario introdurre il tema degli eventi.

Definita la struttura base di un'applicazione, possiamo specificare che entrambe le classi View e Model implementano lo stesso meccanismo di scambio di eventi, ovvero una struttura basata su Publish/Subscribe. La gestione di tali eventi viene lasciata a Croquet in base al tipo e alla classe dalla quale viene generato.

Nello specifico, possiamo identificare quattro tipologie di eventi:

- **Evento di Input**: pubblicato da una View e ricevuto da ogni replica del Model. Nello specifico, si tratta degli eventi inviati attraverso i Reflector e al Model corrente.
- **Evento di Output**: pubblicato da un Model e intercettato da un View. Si tratta quindi degli eventi, precedentemente introdotti, che notificano la vista di un aggiornamento nella classe del modello.

- **Evento del Model**: pubblicato da un Model e intercettato da un altro Model. Questa tipologia di evento è prettamente locale e paragonabile all'esecuzione di un metodo della classe Model.

- **Evento della View**: pubblicato da una View e intercettato da una View. Questo tipo di evento è prettamente locale e corrisponde come per la classe Model al lancio di un metodo.

1.3.6 Microverse

Avendo esposto le nozioni necessarie alla comprensione di Croquet, si ritiene opportuno introdurre il concetto di Microverso attraverso la piattaforma di sviluppo **Microverse World Builder** fornita da Croquet. Questa tematica verrà poi successivamente approfondita anche nei prossimi capitoli.

Per Microverso si intende un mondo condiviso di Metaverso, implementato attraverso un sistema di gestione entità chiamato **Worldcore**. Parleremo quindi di un vero e proprio mondo 3D virtuale, popolato da oggetti tridimensionali con proprietà dinamiche, sincronizzato attraverso Croquet e accessibile attraverso un qualsiasi browser. Inoltre essendo utilizzato come piattaforma di Metaverso presenta un simulatore di leggi fisiche **Rapier Physics Engine**, ed è in grado di renderizzare gli elementi grafici per l'interfaccia web attraverso la libreria **Three.js** [17].

Worldcore

Il Worldcore è un sistema di gestione entità che si trova a un livello superiore di Croquet, con l'obiettivo di semplificare la gestione di grandi quantità di oggetti 3D in un contesto multiutente, utilizzando un sistema basato su actor/pawn che possono essere estesi in modo modulare con dei Mixin [5].

Actor : è un oggetto utilizzato per la gestione della simulazione. Una volta istanziato, viene automaticamente creato il Pawn dedicato a esso, che verrà poi eliminato in caso di distruzione dell'actor. Questo accoppiamento di classi garantisce una comunicazione tra le due parti. A differenza dei Pawn, questa classe è l'unica a essere sincronizzata tra i client.

Pawn : viene generato nella View ogni qualvolta si crei un'istanza di un Actor, ed è utilizzato per la gestione degli input e output.

Mixins : sono funzioni che permettono l'estensione modulare di Actor e Pawn al fine d'integrare funzionalità. Worldcore presenta un insieme di funzioni

Mixin di default, ad esempio quelle per definire una posizione nello spazio 3D.

Struttura

La struttura di un Microverso estende e gestisce a più alto livello un Worldcore attraverso una composizione di Card e Behavior, mantenendo il sistema di comunicazione a eventi Publish/Subscribe che si trova alla base di Croquet. Possiamo quindi individuare due componenti fondamentali:

Card : rappresenta fondamentalmente ogni tipo di oggetto che viene aggiunto al mondo. In fase di creazione devono essere dichiarati tutti i parametri essenziali che ne andranno a descrivere forma, comportamento, aspetto, specifiche e le Behavior in modo da essere serializzato in JSON. Possiamo identificare due tipologie di Card: CardActor, che identifica la Card nella parte Model di Croquet, e CardPawn, che va a definire la Card nella View.

Le Card, come specificato in precedenza, comunicano attraverso il sistema a eventi, precedentemente introdotto, basato su Publish/Subscribe.

Behavior : rappresenta un' azione che può eseguire la card, permettendo di aggiungere funzionalità a CardActor e quindi integrarsi nel Model o a una CardPawn, andando così ad alterare l'aspetto grafico della card nella View. Ogni Behavior può essere agganciata o rimossa da una card dinamicamente.

Avatar

Nel contesto di Microverso, l'Avatar consiste nella rappresentazione dell'utente che si collega al mondo, permettendogli di muoversi e interagire con esso.

La gestione degli Avatar è identica a quella delle Card. Possiamo identificare un AvatarActor per la gestione della parte model relativa agli Avatar, e AvatarPawn, che va a definire l'aspetto grafico di ognuno. Per quanto riguarda la gestione di eventi, è presente **AvatarEventHandler**, che rappresenta un modulo di Behavior.

Capitolo 2

Croquet per la Mixed Reality condivisa

All'interno di questo secondo capitolo verrà trattato l'obiettivo di questa tesi, spiegando nel dettaglio il caso di studio, le possibili applicazioni, e le scelte effettuate nel corso della progettazione. Infine verrà illustrata la soluzione proposta e la relativa struttura.

2.1 Obiettivo

L' obiettivo di questo specifico caso di studio è quello di realizzare applicazioni di Mixed Reality collaborative in real-time, utilizzando come dispositivo per la realtà mista il visore Hololens 2, mentre come piattaforma di sincronizzazione e collegamento Croquet, portando a poter integrare applicazioni di Mixed Reality in quello che è il sistema di comunicazione complesso e distribuito realizzato da Croquet, potendo usufruire delle garanzie di sincronia multiutente, sicurezza e salvataggio dei dati, utilizzando al pieno le funzionalità messe a disposizione.

Come introdotto nel capitolo precedente realizzare un applicazione di realtà mista condivisa comporta il dover gestire molte situazioni specifiche, che però utilizzando Croquet come middleware possiamo trascurare, o per lo meno in parte.

In questa tesi verrà nello specifico illustrata e trattata una delle possibili modalità di collegamento alla piattaforma Croquet, analizzando la possibilità di una comunicazione costante tra Holoens 2 e la piattaforma, attraverso uno scambio di dati, senza alcuna analisi delle prestazioni, ma basandosi prettamente sull' aspetto di fattibilità.

2.1.1 Progettazione attraverso Croquet

Come descritto nel capitolo precedente la piattaforma Croquet presenta molte agevolazioni, che garantiscono una comunicazione di dati costante e una sincronizzazione multiutente, rimanendo a basse latenze, attraverso un flusso di dati sicuro e costante.

Uno degli aspetti fondamentali che porta alla scelta di Croquet, è il continuo salvataggio dei dati attraverso Snapshot che garantisce una maggiore solidità e prevenzione in caso di disconnessioni o problemi di rete. Fondamentale è la struttura delle applicazioni dedicate alla piattaforma, che permette di avere un ampia libertà di sviluppo incentrandosi sulla gestione di eventi e dati nella modalità più generica possibile, permettendo quindi l'adattamento di molteplici applicazioni senza mai gestire alcun codice lato server. In generale possiamo identificare nella piattaforma Croquet tutti i presupposti per ospitare un applicativo di Mixed Reality condivisa, dando la possibilità di accesso anche ad applicazioni secondarie che non vengano forzatamente eseguite in un visore per la realtà mista.

L' utilizzo della piattaforma Croquet come infrastruttura di comunicazione per l' applicazione di Mixed Reality porta in ogni caso a prendere delle scelte implementative, come descritto in precedenza per garantire il collegamento a una sessione di Croquet, ogni client deve eseguire un'istanza di essa, avendo quindi un client Croquet per ogni utente che vuole accedere all' esperienza, ciò implica che Croquet in questo caso venga utilizzato solamente per la gestione dello scambio di dati sincrono multiutente, lasciando la gestione dell' intera scena all' applicativo, naturalmente per far si che il tutto funzioni al meglio, la gestione delle variabili condivise deve essere gestita all'interno del Model Croquet.

2.1.2 Progettazione attraverso Microverso

Introdotte le motivazioni che portano alla scelta di Croquet, ritengo importante esplicitare anche la possibile idea di passare a un concetto di più alto livello, ovvero la possibilità di ospitare un'applicazione di realtà mista all'interno di un vero e proprio Microverso, che permetterebbe di rendere ancora più concreto e immersivo l'obiettivo di questo caso di studio. In questo specifico caso verrebbe sì utilizzata l'infrastruttura sicura e solida realizzata da Croquet, ma naturalmente integrata a tutta la potenzialità di un vero e proprio mondo di Metaverso, con la possibilità di replicare graficamente l'applicazione di realtà mista attraverso la complessa gestione di modelli grafici messa a disposizione, concretizzando quelli che sono gli aspetti di un applicazione di Mixed Reality.

L' utilizzo di un Microverso per la gestione dell' esperienza condivisa, come si può immaginare, porta a dover tener conto di alcuni aspetti aggiuntivi rispetto al solo utilizzo di Croquet. Le funzionalità introdotte in un Microverso sono molteplici, al suo interno vengono gestiti direttamente modelli tridimensionali che possono avere un comportamento specifico e generare eventi, ogni utente collegato viene rappresentato da un'Avatar anch'esso capace di generare eventi, ed è possibile integrare ai modelli 3D proprietà fisiche, alla luce di queste funzionalità molto simili a quelle base messe a disposizione da Unity e MRTK 2 per la realizzazione di un progetto di Mixed Reality, è doveroso chiedersi se la gestione dei modelli tridimensionali debba essere lasciata al Microverso o all' applicazione. La scelta che risulta più appropriata è lasciare la gestione degli oggetti tridimensionali al Microverso, così da avere una più semplice gestione del mondo. Basti pensare a un utente che voglia partecipare alla condivisione attraverso un browser Web e non un visore, avrebbe le stesse possibilità di interazione generando lo stesso tipo di eventi che verranno poi replicati agli utenti provvisti di visori.

2.1.3 Scelta di implementazione

Una volta esposte le due tipologie di progetti che ci si presentano, è necessario analizzare nello specifico le due scelte. La possibilità di ospitare un' applicazione di realtà mista in un mondo dedicato come il Microverso è sicuramente l'idea più allettante, ma non la scelta ideale per questo caso di studio.

In questa tesi viene mostrata una possibile implementazione di un applicazione di realtà mista condivisa gestita tramite Croquet, la scelta di tale piattaforma rispetto un Microverso è dovuta, prima di tutto, dal fatto che, alla base di un Microverso risiede comunque un' architettura Croquet, si è preferito un approccio a step partendo dalla gestione a più basso livello, questa scelta è motivata anche dal fatto che per interfacciarsi alla piattaforma Croquet è stato utilizzato il progetto di tesi **croquet-proxy** sviluppato da Mazzoli Alessandro che verrà introdotto successivamente. In ogni caso la possibilità di implementazione utilizzando un Microverso è stata presa in considerazione ed esplorata, ma purtroppo a causa del tempo limitato a disposizione e la gestione differente a livello di struttura fisica del mondo, visto che si trova ancora in fase di BETA, avrebbe comportato la riscrittura dell'intero proxy di comunicazione ed è stata momentaneamente scartata.

2.1.4 Idea

Osservato l'obiettivo di questa tesi e la scelta implementativa presa, è necessario specificare l'idea di base sulla quale si incentra il progetto. Presa la decisione quindi di utilizzare Croquet come base per l'implementazione dell' esperienza condivisa, prima di tutto è necessario avere una panoramica generale del tipo di esperienza che si vuole andare a generare, visto le potenzialità di Croquet possiamo affermare di non dover gestire alcun codice lato server, come specificato nello scorso capitolo, ponendo l'attenzione solamente sulla parte relativa al client. L'idea è perciò quella di realizzare un unico client composto da Model e View necessari a Croquet per il collegamento, e l'applicazione di Mixed Reality da condividere, naturalemnte apportando le modifiche necessarie a entrambe le parti, per garantire la ricezione e l'invio di eventi alla rete di Reflector e applicare poi tali cambiamenti a tutti i client connessi.

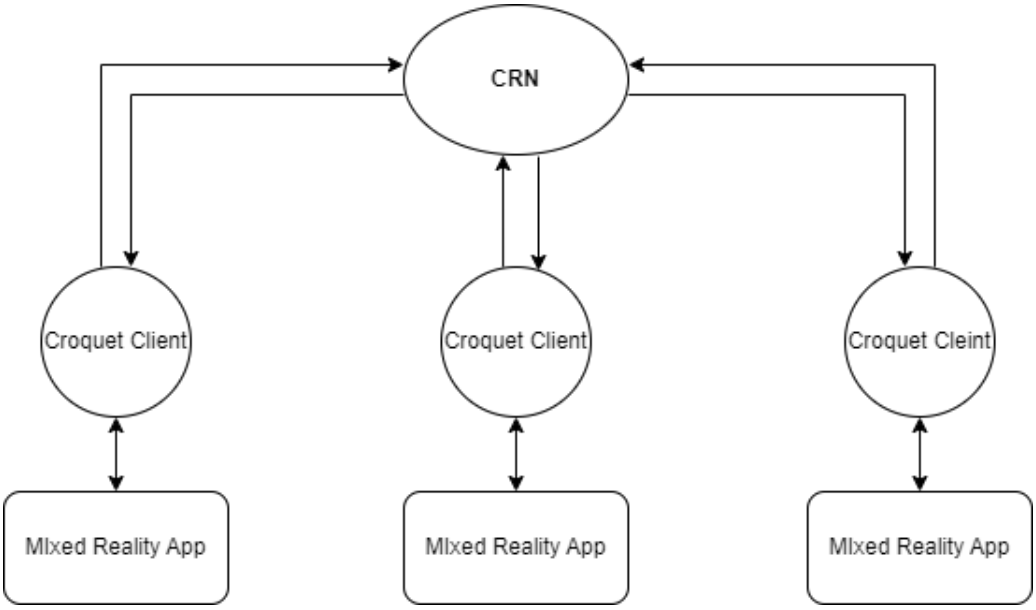

Figura 2.1: Rappresentazione dell'idea

2.1.5 Dispositivo utilizzato e piattaforme di sviluppo

In questo specifico caso di studio, il dispositivo utilizzato per realizzare il progetto di Mixed Reality è Hololens 2.
Holoens 2 è un wearable computer con sistema operativo Windows Holographic basato su Windows 10, sviluppato da Microsoft e rilasciato nel 2019. Il dispositivo è dotato di lenti olografiche trasparenti e di una serie di sensori e telecamere, i quali consentono il tracciamento del movimento dell'utente e il

riconoscimento di gesture delle mani con relativo posizionamento e distinzione di mano destra e sinistra.

Nel visore è presente un sistema di tracciamento oculare che permette di gestire delle funzionalità tramite il movimento degli occhi, e un sistema audio integrato che permette di ascoltare suoni e interagire attraverso comandi vocali [7].

Unity

Lo sviluppo di software dedicati alla realtà mista è stato possibile grazie all'utilizzo del motore grafico 3D e 2D Unity.
Questo motore grafico è in costante evoluzione e fornisce supporto per lo sviluppo di grafica 2D e 3D, offrendo un set di strumenti robusto per la gestione di materiali, shader, illuminazione e fisica.

Unity presenta una architettura dei progetti abbastanza semplice, dividendo la struttura in più scene, ognuna delle quali conterrà tutti gli oggetti necessari per l'utilizzo del livello. Essa tratta la gestione degli oggetti grafici attraverso relazioni genitore-figlio e rende semplice l'integrazione di oggetti secondari e proprietà.

Il motore grafico, oltre ad avere una gestione semplificata di oggetti e un'architettura semplice, è sostenuto da una potente API di scripting che offre l'accesso a funzionalità comuni e specifiche.

Un aspetto fondamentale è il supporto di build multi piattaforma, che permette la creazione di progetti per un numero immenso di piattaforme installando il kit di supporto dedicato. Nell'attuale caso di studio il progetto Unity è stato configurato attraverso MRTK 2 che verrà esposto successivamente, permettendo di realizzare applicazioni per la realtà mista, e dando un supporto alla gestione dell'architettura del progetto configurandolo appositamente per il visore, al fine di realizzare un progetto che interpreti e rielabori i modelli 3D della scena sotto forma di ologrammi.

MRTK 2

MRTK 2 o Mixed Reality ToolKit 2 è un tool sviluppato da Microsoft che permette una configurazione rapida e semplice di un progetto Unity 3D per la

Mixed Reality.

Nello specifico, il tool permette di gestire i vari pacchetti da integrare al progetto, configurandolo successivamente per poter gestire tutti gli aspetti fondamentali della realtà mista. Fornisce inoltre il supporto per un sistema di input multi piattaforma, l'integrazione di modelli grafici base per interazioni spaziali e interfacce grafiche, ammette la realizzazione di prototipi attraverso un sistema di simulazione e soprattutto garantisce il supporto per un'ampia gamma di dispositivi. [1].

Open XR

OpenXR è una API standard open royalty free, realizzata da The Khronos Group, che fornisce l'accesso a una vasta gamma di dispositivi nello spettro della realtà mista.

L'obiettivo di Open XR è quello di realizzare un' interfaccia standard per la creazione di applicazioni dedicate alla Extended Reality, permettendo lo sviluppo su differenti piattaforme hardware senza la necessità di scrivere codice specifico per ogni dispositivo o sistema operativo. Ciò rende la creazione di applicazioni più efficiente e semplificata per gli sviluppatori, e consente ai consumatori di utilizzare le applicazioni su un'ampia gamma di dispositivi [13].

2.2 Progettazione del client

Analizzato nello specifico il lato client si possono individuare due componenti fondamentali: l' applicativo realizzato in Unity con il supporto di MRTK 2 e quello che si può definire il client Croquet. A questo punto è necessario definire due aspetti principali, ovvero, prima di tutto come avviene la comunicazione tra il client Croquet (scritto in JavaScript) e l' applicazione Unity e per secondo come viene gestita la scena condivisa.

2.2.1 croquet-proxy

Per quanto riguarda la questione di collegamento tra le due parti è stato preso in considerazione, come introdotto in precedenza, l'utilizzo del proxy realizzato da Mazzoli Alessandro, che permette di stabilire una comunicazione multiutente e costante a un qualsiasi client Croquet, dando la possibilità di gestire il Model e la View dedicate alla sessione di collegamento, in modo da garantire l'accesso alle potenzialità della piattaforma a una estesa gamma di linguaggi. Il proxy permette quindi l'ascolto e l'invio degli eventi di Croquet attraverso un client Socket.io e uno scambio costante di JSON [22].

2.2.2 Gestione della scena

Una volta determinato come far comunicare le due parti che andranno a comporre il client, è necessario stabilire quali limiti comporta la scelta di tale implementazione, come è stato accurato la gestione della scena condivisa in questo caso è lasciata all'applicazione scritta in Unity, tale scelta comporta perciò il dover utilizzare Croquet come piattaforma di scambio dati.

Per spiegare al meglio lo scenario che si viene a creare è propedeutico pensare a un esempio, si metta caso che l'applicazione di realtà mista comporti la sola visualizzazione nello spazio di un cubo, per far sì che la posizione dell'oggetto venga condivisa con tutti i client connessi è necessario che le tre variabili cartesiane X, Y e Z, che identificano la posizione del cubo nello spazio, vengano condivise, ciò comporta la realizzazione nel Model di Croquet di tre variabili per tenere traccia appunto di tali valori e condividerli al sistema di Reflector. Questa tipologia di approccio è necessaria al fine di realizzare l'esperienza condivisa, come si può notare presenta diverse limitazioni e accortezze per la realizzazione del progetto, ma risulta fondamentale alla creazione di sistemi molto più complessi in futuro.

2.2.3 Possibilità di connessione a Hololens e problematiche

Un aspetto importante da tenere in considerazione è la possibilità di collegamento a Hololens, una delle problematiche affrontate appunto è stata la realizzazione del client per l'allaccio di Hololens al croquet-proxy.

Nello specifico un progetto Unity integrato a MRTK 2 per la realizzazione di un'applicazione di realtà mista, viene configurato e compilato come un'applicazione per UWP, ovvero, Universal Windows Platform, cioè una piattaforma di sviluppo di applicazioni introdotta da Microsoft per consentire agli sviluppatori di creare applicazioni eseguibili su diversi dispositivi Windows, come PC, tablet, smartphone, Xbox e HoloLens. Realizzare un'applicazione che venga compilata per UWP integra diverse limitazioni visto che comporta l'utilizzo un set di API e librerie comuni chiamate Windows Runtime (WinRT), che consentono alle applicazioni di funzionare su diverse architetture di processori e sistemi operativi Windows.

Al fronte delle limitazioni derivanti dall'UWP, è sorta la prima considerevole problematica, ovvero realizzare un client che consenta la connessione a Socket.io, ma che utilizzi le librerie consentite. Effettuate le dovute ricerche, e testando molteplici soluzioni, si è giunti a conclusione che non è possibile realizzare un client UWP che riesca a garantire una connessione a un server Socket.io, ciò è dovuto dal fatto che alla base di Socket.io viene utilizzata la libreria **WebSockets**, che effettivamente è presente anche all'interno dell'UWP, ma come sostenuto dalla documentazione di Socket.io

> Socket.IO is **NOT** a WebSocket implementation. [21]

Ovvero Socket.io utilizza WebSockets per il trasporto se possibile, aggiungendo dei metadati a ogni pacchetto, definendo che un client realizzato tramite WebSocket non sarebbe del tutto in grado di collegarsi a un server Socket.io.

2.3 Bridge

La necessità di stabilire un collegamento tra il croquet-proxy e l'applicazione di mixed reality, al fronte della problematica esposta precedentemente, ha portato a dover introdurre una terza parte che permettesse l'aggancio del proxy alla applicazione, è stato realizzato una sorta di ponte, che garantisca il collegamento e il passaggio di eventi tra il server Socket.io presente nel proxy e il client presente nell'applicazione.

L'idea che sta alla base di questo ponte di comunicazione è quindi la realizzazione di un client Socket.io, e un server compatibile con le librerie dell'UWP, garantendo che gli eventi intercettati da Croquet vengano tradotti e inviati nel

formato richiesto alla applicazione di Mixed Reality e viceversa, con il solo scopo di ricreare e reindirizzare gli eventi. Il bridge di comunicazione, una volta avviato, si collegherà al croquet-proxy tramite il client al suo interno, e renderà disponibile un server alla quale l'applicazione di Mixed Reality si potrà collegare, completando così il flusso di comunicazione.

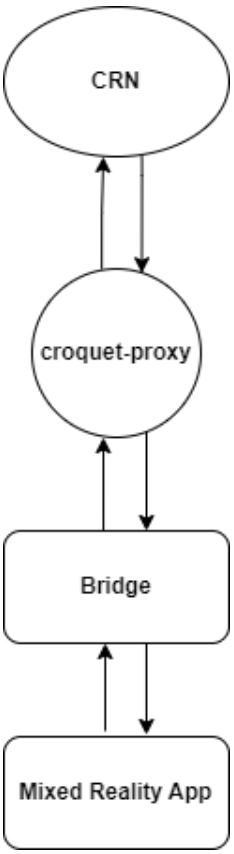

Figura 2.2: Rappresentazione dell'idea integrando il bridge

2.3.1 Connessione al croquet-proxy

Risolto il problema di compatibilità e connessione, è necessario analizzare gli eventi generati dal croquet-proxy utilizzati per il corretto collegamento e funzionamento dell'intera infrastruttura realizzata. Possono essere identificate due tipologie principali di eventi, verranno perciò suddivisi in due parti, ovvero: eventi di Croquet ed eventi del proxy.

Eventi di Croquet

Per eventi di Croquet si intende quella serie di eventi che permettono di interfacciarsi al sistema di Croquet, potendo così gestire le subscription e le publish.

- **subscribe:** Il proxy è in grado di riceve eventi da parte del client Socket.io che permettono la subscription a un determinato scope ed event.
- **unsubscribe:** Il proxy gestisce il caso di cancellazione della subscription.
- **publish:** Il proxy è in grado di intercettare e replicare gli eventi di publish che vengono ricevuti tramite i client Socket.io.
- **event:** In caso di ricezione di un evento di publish, il proxy emette tale evento ai client che hanno effettuato la subscription ai relativi scope ed event, replicando il comportamento di Croquet.

Eventi del proxy

Per eventi del proxy, si intende la serie di eventi che non replicano il comportamento di Croquet, ma vengono utilizzati a scopo di configurazione e gestione della connessione multiutente e dei dati. Successivamente verranno descritti gli eventi necessari per stabilire una connessione con il proxy adattati attraverso il bridge.

- **connection-ready:** L'evento in questione viene rilasciato dal proxy verso il client per notificare l'avvenuta connessione, richiedendo di poter procedere con la realizzazione del socket.
- **join:** Questo specifico evento è emesso dal client come risposta al "connection-ready" e notifica la possibilità lato client di realizzare il socket.
- **ready:** Questo evento è emesso dal proxy verso il client e notifica la compiuta realizzazione del socket dando il via alla comunicazione stessa. Al fronte della ricezione di tale evento il client potrà ora procedere con l'emissione delle subscription e dei publish.

Una volta elencati gli eventi di configurazione personalizzati, e quindi non generati di default da Socket.io, è necessario descrivere altri due eventi specifici realizzati per la notifica dei dati e il loro cambiamento.

- **data:** L'evento in questione viene emesso dal proxy in fase di creazione del socket, e viene utilizzato per notificare al client il valore dei dati condivisi nello stato corrente in cui si effettua la connessione.

- **data-update:** Questo specifico evento viene utilizzato per notificare il cambiamento dei dati nel Model di Croquet, è stato momentaneamente silenziato lato Bridge perché ritenuto eccessivo visto che la notifica dei dati viene già effettuata tramite l'evento "event" precedentemente esposto.

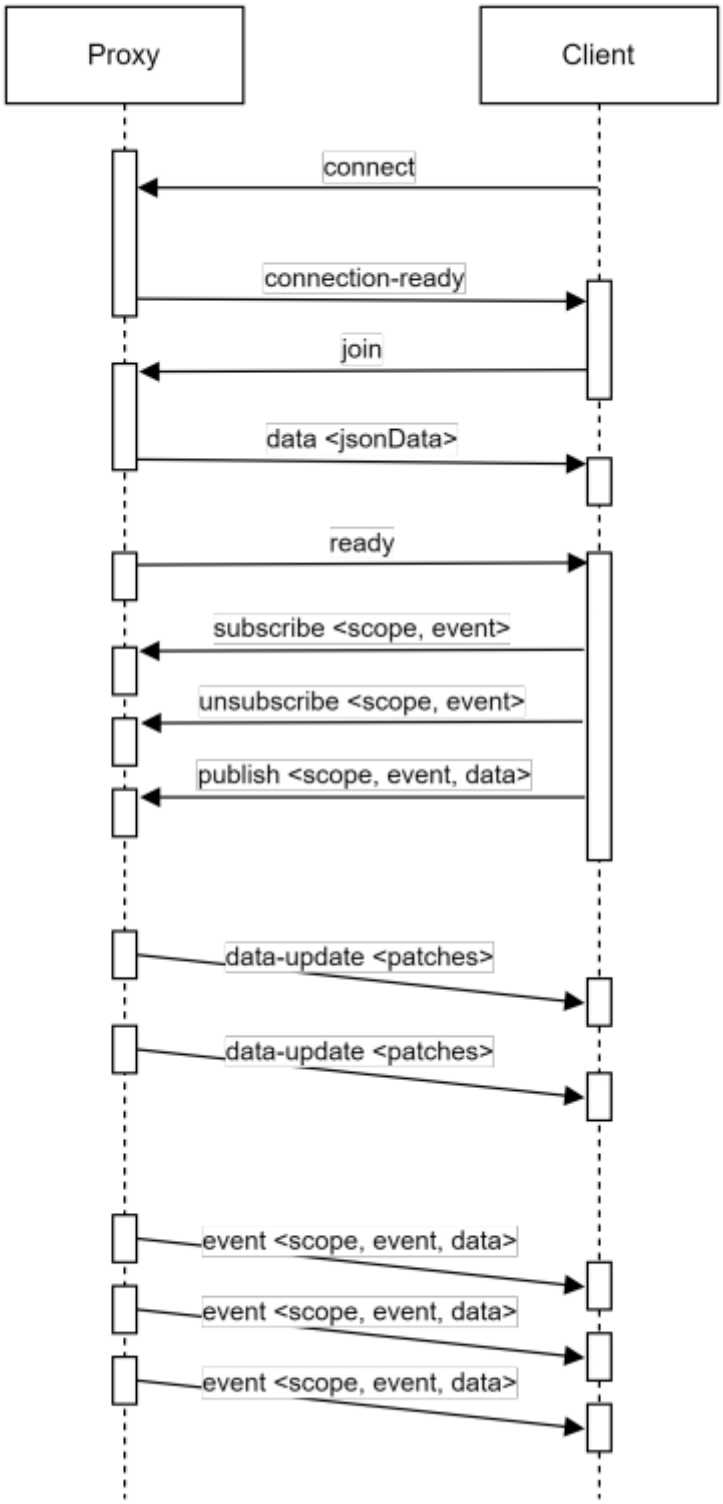

Figura 2.3: Flusso di comunicazione degli eventi [22]

2.4 Progettazione dell'applicazione di Mixed Reality

Una volta definita l'dea che sta alla base del ponte di comunicazione, è necessario introdurre la struttura che andrà a comporre la base dell'applicazione di Mixed Reality.

L'applicativo è stato realizzato basandosi sulla struttura definita da Mazzoli Alessandro nella tesi relativa al croquet-proxy, ed è stata adattata alla struttura di un progetto Unity sviluppato in C#.

Il client è perciò composto da una struttura che rispecchia la base di Croquet, ed è possibile identificare un Model per la gestione dei dati che andrà a contenere e definire le variabili da condividere, proprio come avviene lato Croquet e una View, che permetterà la gestione degli eventi di sottoscrizione e pubblicazione con i relativi metodi da eseguire in caso di ricezione di publish; questa specifica parte del client, permetterà quindi la gestione degli eventi e dei dati, essendo la parte da modificare in base alle proprie esigenze. Il secondo componente fondamentale, oltre al client, che gestirà il collegamento al bridge, è lo script main, ovvero una classe che permette di agganciare gli oggetti tridimensionali della scena alla View realizzata e consente la loro gestione.

Capitolo 3

Implementazione del Bridge di comunicazione

In questo terzo capitolo viene descritta l'implementazione del ponte di comunicazione, verrà analizzato nello specifico il codice realizzato e le scelte prese, così da poter avere una panoramica completa dell'intera infrastruttura realizzata.

3.1 Scelte implementative

La realizzazione del ponte di comunicazione ha portato a prendere delle scelte decisive dal punto di vista implementativo, verranno analizzate le scelte che hanno portato alla decisione del linguaggio di programmazione, delle librerie utilizzate e del protocollo lato client.

3.1.1 Protocollo di rete

Al fronte delle problematiche esposte nello scorso capitolo riguardanti le limitazioni dell Universal Windows Platform, è stata presa la decisione di utilizzare come protocollo di rete bridge-client il TCP a basso livello, utilizzato all'interno di altri progetti che permettevano l'aggancio di applicazioni di Mixed Reality a componenti web, avendo così la certezza del funzionamento e dell'integrazione di librerie che ne permettano la connessione e scambio dati all'interno dell'UWP.

3.1.2 Linguaggio di programmazione e Librerie

Il linguaggio di programmazione scelto per la realizzazione del bridge di comunicazione è il JavaScript utilizzando Node.js, nello specifico si è ritenuto

importante mantenere lo stesso del croquet-proxy, visto che si tratta effettivamente di una sua estensione, che permetterà l'aggancio di client realizzati per Universal Windows Platform, così da garantire l'esecuzione del client e del proxy sulla stessa macchina evitando eccessivi cambiamenti e limitando le possibili latenze.

Determinato il linguaggio di programmazione da utilizzare, ritengo necessario introdurre le librerie utilizzate per la realizzazione del ponte, nello specifico, il client per l'aggancio al croquet proxy è stato realizzato attraverso il modulo **socket.io-client**, come previsto per il corretto collegamento, mentre il server TCP a basso livello è stato realizzato attraverso il modulo **net**, così da avere una certezza dal punto di vista di compatibilità.

Node.js

Node.js è un ambiente di esecuzione JavaScript lato server che consente di eseguire codice JavaScript al suo interno, anziché solo all'interno di un browser. In altre parole, Node.js consente di utilizzare JavaScript per creare applicazioni web lato server.

Node.js è costruito sulla piattaforma V8 JavaScript Engine di Google, che è in grado di eseguire codice JavaScript molto rapidamente. Ciò rende Node.js particolarmente utile per la gestione di applicazioni web ad alta intensità di elaborazione, come le applicazioni di streaming o le applicazioni che richiedono la gestione di molte richieste simultanee.

Inoltre, Node.js fornisce anche una vasta libreria di moduli e strumenti che semplificano lo sviluppo di applicazioni web, consentendo agli sviluppatori di concentrarsi sulla logica dell'applicazione anziché sulla gestione delle funzionalità di basso livello.[10].

socket.io-client

Questo specifico modulo viene utilizzato per l'integrazione in Node.js della libreria client di Socket.io, in particolare, Socket.IO permette di gestire le connessioni in tempo reale tra client e server tramite WebSockets, in modo da consentire l'invio e la ricezione di dati in modo asincrono e in tempo reale. Socket.IO offre anche la possibilità di utilizzare altri protocolli di trasporto, come ad esempio Server-Sent Events e polling XHR, in base alle capacità dei client e del server.

net

Net identifica lo specifico modulo Node.js che fornisce una serie di API per la creazione di server e client TCP (Transmission Control Protocol) e per la

manipolazione di socket di rete.

Utilizzando il modulo net è possibile creare server TCP altamente scalabili e performanti, che possono gestire numerose connessioni simultanee, questo lo rende particolarmente utile per applicazioni real-time. Ciò significa che è possibile creare un server TCP personalizzato che può gestire richieste e inviare risposte ai client, senza dover utilizzare HTTP o altri protocolli a più alto livello.

3.1.3 Struttura del programma

Il software realizzato è stato pensato come un unico file JavaScript, contenete quindi un client Socket.io e un server TCP, gli eventi di uno vengono agganciati all'altro, e i pacchetti di dati intercettati vengono scomposti e ricomposti (trattandosi di JSON), per la corretta comunicazione.

Nello specifico oltre a garantire il passaggio e la traduzione dei dati sono stati integrati dei meccanismi, descritti poi in seguito, che permettano la corretta sincronizzazione e, in fase di chiusura, il corretto scollegamento del client.

Verranno successivamente descritti separatamente il client Socket.io, il server TCP e infine gli eventuali meccanismi implementati per una gestione ottimale del sistema.

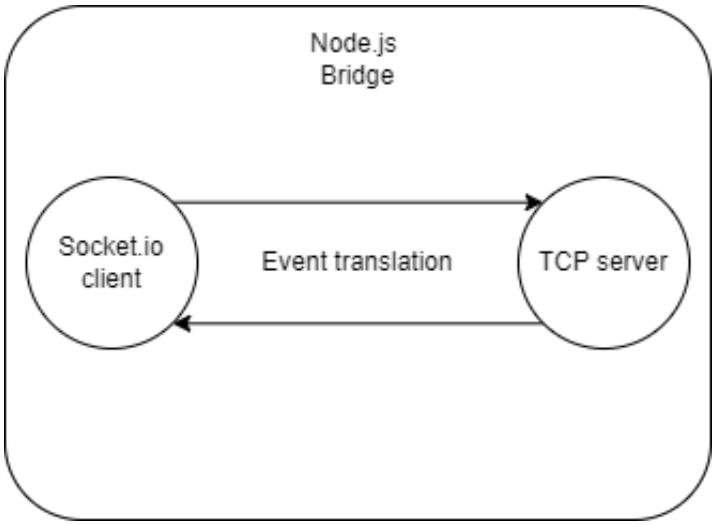

Figura 3.1: Struttura logica del bridge

3.2 Client di Socket.io

In questa specifica sezione si entrerà nel dettaglio del codice realizzato per il lato client Socket.io, analizzando specifici segmenti di codice.

```
//Socket.io client side
const { io } = require("socket.io-client");
let socketIo = io('http://localhost:3000', { reconnect: true });

socketIo.on('connect', function (socket) {
    console.log('Connected!');
});

socketIo.on('event', function (scope, event, data) {
    server.emit('event-croquet', scope, event, data);
});

socketIo.on('data', function (chunk) {
    server.emit('data-croquet', chunk);

});

socketIo.on('data-update', function (chunk) {
    server.emit('data-update-croquet', chunk);
});

socketIo.on("connection-ready", function (chunk) {
    connectionReady = true;
    server.emit('connection-ready');
});

socketIo.on('ready', function (chunk) {
    server.emit('ready');
});
```

Listato 3.1: Sezione di codice relativa all'itera implementazione del client Socket.io

Come è possibile notare la sezione relativa al client di Socket.io oltre a effettuare una connessione al proxy, gestisce gli eventi descritti nel precedente capitolo, nello specifico, ogni evento intercettato viene replicato al server TCP, così da garantirne l'invio al client collegato.

3.3 Server TCP

In questa sezione verrà analizzato il codice relativo all'implementazione del server TCP, verrà descritta nel particolare la strategia di ricezione, invio e replicazione degli eventi.

```
//TCP server side
const Net = require('net');
const port = 10000;
const server = new Net.createServer();

server.listen(port, () => {
    console.log(`Server listening for connection requests on socket
        localhost:${port}`);
});
```

Listato 3.2: Sezione di codice relativa all'inizializzazione del server

Addentrandosi nello specifico dell'implementazione, si può notare la gestione di due tipologie differenti di eventi, i primi sono gli eventi ricevuti dal socket di comunicazione, cioè quelli che andranno a identificare i pacchetti ricevuti dal client (applicazione di Mixed Reality) e che dovranno essere tradotti e inviati al croquet-proxy.

```
server.on('connection', function (socket) {

    ...

    socket.on('data', function (chunk) {
        let splitString = chunk.toString().split('\n');
        splitString.forEach((val) => {
            if (val !== '') {
                try {
                    obj = JSON.parse(val);
                    switch (obj.action) {
                        case 'subscribe':
                            subscriptionList.push({ scope:
                                obj.scope.trim(), event:
                                obj.event.trim() });
                            socketIo.emit('subscribe', scope =
                                obj.scope.trim(), event =
                                obj.event.trim());
                            break;
                        case 'publish':
```

```
                        socketIo.emit('publish', scope =
                            obj.scope.trim(), event =
                            obj.event.trim(), data = obj.data);
                        break;
                    case 'join':
                        socketIo.emit('join');
                        break;
                    case 'unsubscribe':
                        subscriptionList.splice(subscriptionList
                            .indexOf({ scope: obj.scope.trim(),
                                event: obj.event.trim() }), 1);
                        socketIo.emit('unsubscribe', scope =
                            obj.scope.trim(), event =
                            obj.event.trim());
                }
            } catch (e) { }
        }
    });
});

...

}
```

Listato 3.3: Sezione di codice relativa agli eventi ricevuti dall'applicazione di Mixed Reality

La seconda serie di eventi gestita riguarda l'insieme di eventi descritti nella sezione relativa al client di Socket.io, che dovranno essere emessi attraverso il socket di comunicazione al client TCP.

```
...

    socket.on('end', function () {
        console.log('Closing connection with the client');
        subscriptionList.forEach(function (element) {
            socketIo.emit('unsubscribe', scope = element.scope, event
                = element.event);
        });
        subscriptionList = [];
        socket.destroy();
        socketIo.disconnect();
        socketIo.connect();
    });
```

```
    socket.on('error', function (err) {
        console.log(`Error: ${err}`);
    });

    server.on('ready', function () {
        socket.write(JSON.stringify({ action: 'ready' }) + "\n");
    });

    server.on('connection-ready', function () {
        socket.write(JSON.stringify({ action: 'connection-ready' }) +
            "\n");
    });

    server.on('data-croquet', function (chunk) {
        socket.write(JSON.stringify({ action: 'data', data: chunk })
            + "\n");

    });

    server.on('event-croquet', function (scopeVal, eventVal,
        dataVal) {
        socket.write(JSON.stringify({ action: 'event', scope:
            scopeVal, eventName: eventVal, data: dataVal }) + "\n");
    });

    server.on('data-update-croquet', function (chunk) {
        //socket.write(JSON.stringify({ action: 'data-update', data:
            chunk }));
    });
});
```

Listato 3.4: Sezione di codice relativa agli eventi ricevuti dal client Socket.io

In particolare è necessario introdurre la funzione dell' evento *'data'* [3.3], ovvero l'evento emesso di default dal socket di comunicazione in caso di ricezione di dati. Il pacchetto ricevuto da questo evento conterrà quindi le specifiche in formato JSON per ogni evento da replicare, ogni set di specifiche è delimitato dal carattere terminatore **'\n'**. In questa specifica sezione è stato necessario integrare, anche se in modo spartano e non ottimizzato, ma comunque propedeutico al fine del caso di studio, una scomposizione del pacchetto ricevuto in più JSON, ciò si è reso necessario per catturare tutti gli eventi emessi dal client viste le alte frequenze di aggiornamento dell' applicativo di Mixed Reality, che portavano a saturare il socket mettendo in coda più JSON in un unico pac-

chetto, o in alcune casistiche, inviando dati parziali. Scomposto ogni pacchetto viene definito l'evento da realizzare e inoltrare al croquet-proxy.

Altra parte fondamentale, e degna di una descrizione, è l'vento generato automaticamente in caso di disconnessione del client TCP, ovvero **'end'** [3.4], per gestire l'avvenuta disconnessione del client e garantire che il croquet-proxy venga aggiornato a riguardo, oltre alla distruzione del socket TCP e a ricreare il socket di connessione con il croquet-proxy, vengono emessi gli eventi di **'unsubscribe'** per evitare la continua emissione di publish inutili da parte di Croquet. Ogni **'subscription'** effettuata viene quindi salvata all'interno di un array di dictionary così da tenerne conto in caso di disconnessioni come esposto precedentemente.

Per il corretto funzionamento del bridge è stata implementata una variabile booleana (connectionReady) che tiene traccia della ricezione dell' evento **'connection-ready'**, ciò è necessario nel caso in cui il bridge venga eseguito prima del client TCP, così da poter comunicare al client in fase di connessione, la possibilità di procedere.

Capitolo 4

Implementazione del Client

All'interno di questo quarto capitolo verrà trattata la progettazione e l'analisi del codice dell'applicazione di Mixed Reality, e verranno illustrate le scelte implementative prese, nel corso del capitolo verranno perciò illustrate le classi ritenute essenziali che compongono l'infrastruttura del client.

Come esplicitato nel secondo capitolo, la struttura del client rispecchia quella realizzata da Mazzoli Alessandro nel corso del suo progetto, tale struttura è stata riadattata al linguaggio C# per integrarla sotto forma di script nell'applicazione.

4.1 Struttura del client

Prima di passare alla parte vera e propria di implementazione ritengo necessario parlare di come è stato strutturato il client. Naturalmente l'applicazione realizzata in Unity, una volta integrata con il toolkit MRTK 2, varia la propria struttura, ma la gestione degli oggetti e l'aggancio di script a essi rimate totalmente identica. Il client realizzato avrà quindi una classe "main", che come di norma, estende la classe **MonoBehaviour** così da garantire l'integrazione dello script a un elemento grafico, potendo procedere con l'esecuzione.

Possiamo quindi identificare nel progetto oltre alla classe main, un client TCP che permette la ricezione e l'invio di eventi, e la replicazione delle due classi View e Model di Croquet che permetteranno di gestire i dati condivisi e i relativi eventi da emettere.

4.1.1 Librerie utilizzate

Come accennato in precedenza, ed è semplice intuire, date le restrizioni che comporta la build del progetto in Universal Windows Platform, non sono state utilizzate librerie esterne, ma solamente quelle messe a disposizione dal

sistema, in particolare ritengo opportuno citare **System.Net.Sockets**, ovvero la libreria che ha permesso l'implementazione del client TCP potendo perciò effettuare la connessione al bridge senza alcun errore.

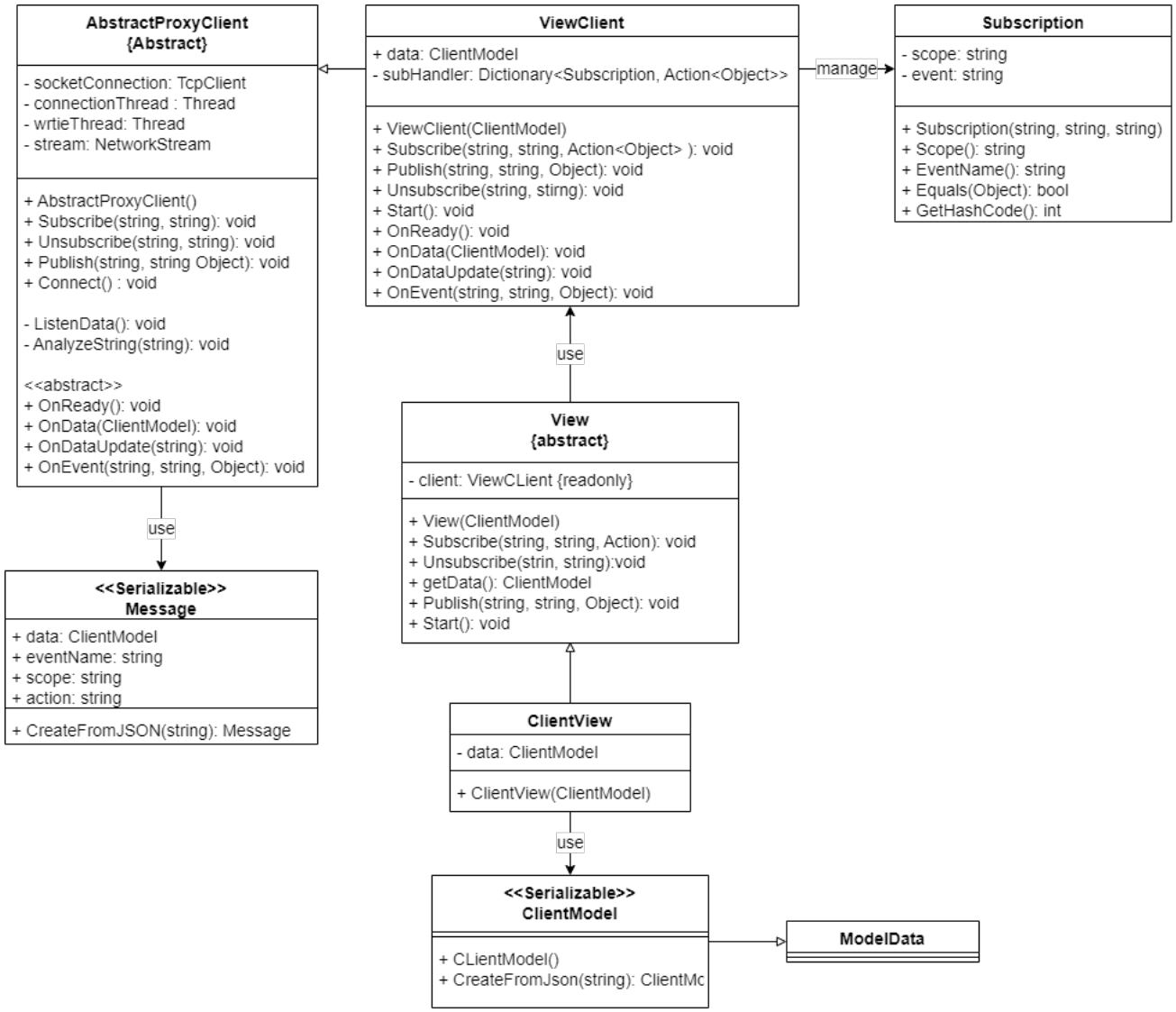

Figura 4.1: Diagramma delle classi che compongono il client

4.2 AbstractProxyClient

All'interno di questa classe viene gestita l'intera connessione con il server TCP e la gestione dei vari eventi. Come si può notare, anche lato client, è stata implementata la separazione dei pacchetti in più JSON nel caso in cui vengano accumulati, così da non perdere alcun evento.

```
...

    private void AnalyzeString(string val)
    {
        string[] str = val.Split('\n');
        foreach (string splitted in str)
        {
            if (splitted.Length > 0)
            {
                Message message = Message.CreateFromJSON(splitted);
                switch (message.action)
                {
                    case "connection-ready":
                        this.Connect();
                        break;
                    case "ready":
                        this.OnReady();
                        break;
                    case "data":
                        Debug.Log("data");
                        this.OnData(message.data);
                        break;
                    case "event":
                        this.OnEvent(message.scope, message.eventName,
                            message.data);
                        break;
                }
            }
        }
    }

...
```

Listato 4.1: Frammento di codice che mostra la separazione dei JSON e la corretta intercettazione dell'evento

Ogni evento noto è stato implementato ricreando il relativo metodo, così da replicare gli eventi precedentemente descritti. Il client è quindi in grado di generare la stringa JSON che andrà a definire l'evento, serializzando dei dati al suo interno, se necessario, e invierà l'evento generato al server TCP.

```
...

public void Subscribe(string scope, string eventName)
{
    if (stream.CanWrite)
    {
        string clientMessage = "{\"action\":\"subscribe\",
            \"event\":\"" + eventName + " \", \"scope\":\"" +
            scope + " \"}" + '\n';
        byte[] clientMessageAsByteArray =
            Encoding.ASCII.GetBytes(clientMessage);
        stream.WriteAsync(clientMessageAsByteArray, 0,
            clientMessageAsByteArray.Length);
    }
}

public void Unsubscribe(string scope, string eventName)
{
    if (stream.CanWrite)
    {
        string clientMessage = "{\"action\":\"unsubscribe\",
            \"event\":\"" + eventName + " \", \"scope\":\"" +
            scope + " \"}" + '\n';
        byte[] clientMessageAsByteArray =
            Encoding.ASCII.GetBytes(clientMessage);
        stream.WriteAsync(clientMessageAsByteArray, 0,
            clientMessageAsByteArray.Length);
    }

}

public void Publish(string scope, string eventName,
    System.Object data)
{
    if (stream.CanWrite)
    {
        string clientMessage = "{\"action\":\"publish\",
            \"event\":\"" + eventName + " \", \"scope\":\"" +
            scope + " \", \"data\":" + JsonUtility.ToJson(data) +
```

```
            " }" + '\n';
        byte[] clientMessageAsByteArray =
            Encoding.ASCII.GetBytes(clientMessage);
        stream.WriteAsync(clientMessageAsByteArray, 0,
            clientMessageAsByteArray.Length);

    }

}

public void Connect()
{
    if (stream.CanWrite)
    {
        string clientMessage = "{\"action\":\"join\"}" + '\n';
        byte[] clientMessageAsByteArray =
            Encoding.ASCII.GetBytes(clientMessage);
        stream.WriteAsync(clientMessageAsByteArray, 0,
            clientMessageAsByteArray.Length);
    }
}

public abstract void OnReady();
public abstract void OnData(ClientModel jsonData);
public abstract void OnDataUpdate(string jsonPatch);
public abstract void OnEvent(string scope, string eventName,
    System.Object data);
...
```

Listato 4.2: Frammento di codice relativo all'invio di eventi

In caso in cui un evento venga ricevuto, dopo gli opportuni controlli e separazioni [4.1], viene realizzato un oggetto della classe Message utilizzato per determinare l'azione che si vuole eseguire, e passare al relativo metodo i dati necessari all'esecuzione.

```
[System.Serializable]
public class Message
{
    public ClientModel data;
    public string eventName = "";
    public string scope = "";
    public string action = "";
```

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

Listato 4.3: Rappresentazione della classe Message

4.2.1 ViewClient

La classe in questione estende AbstractProxyClient e ne determina i metodi astratti.

```
    ...

    public override void OnReady()
    {
        foreach (KeyValuePair<Subscription, Action<System.Object>>
            entry in this.subHandler)
        {
            base.Subscribe(entry.Key.Scope(), entry.Key.EventName());
        }
    }

    public override void OnData(ClientModel jsonData)
    {
        this.data = jsonData;

    }

    public override void OnDataUpdate(string jsonData)
    {
        JsonUtility.FromJsonOverwrite(jsonData, this.data);
    }

    public override void OnEvent(string scope, string eventName,
        System.Object data)
    {
        Action<System.Object> tmp;
        if (this.subHandler.TryGetValue(new
            Subscription(scope.Trim(), eventName.Trim()), out tmp))
        {
            tmp(data);
```

```
        }
    }

    ...
```

Listato 4.4: Definizione dei metodi astratti della classe AbstractProxyClient

Questa classe può essere vista come una View del client TCP, come si intuisce dal nome, ovvero permette di eseguire i metodi relativi al lancio di eventi, tiene traccia dei metodi da eseguire definiti nelle subscription attraverso il Dictionary subHandler [4.5], esegue tali metodi in caso di ricezione di publish replicando la struttura a eventi Publish/Subscribe di Croquet e gestisce l'aggiornamento dei dati del Model interno tramite la gestione degli eventi 'data' e 'data-update' descritti in precedenza.

```
    ...

    private Dictionary<Subscription, Action<System.Object>>
        subHandler = new Dictionary<Subscription,
        Action<System.Object>>();

    ...

     public void Subscribe(string scope, string eventName,
         Action<System.Object> onEvent)
     {
        this.subHandler.Add(new Subscription(scope.Trim(),
            eventName.Trim()), onEvent);
     }

    public void Unsubscribe(string scope, string eventName)
    {
        base.Unsubscribe(scope, eventName);
        this.subHandler.Remove(new Subscription(scope.Trim(),
            eventName.Trim()));
    }

    ...
```

Listato 4.5: Frammento di codice che definisce la gestione degli eventi Subscribe/Unsubscribe

4.3 ClientModel

```
[System.Serializable]
public class ClientModel : ModelData
{
    public static ClientModel CreateFromJSON(string jsonString)
    {
        return JsonUtility.FromJson<ClientModel>(jsonString);
    }

    public ClientModel()
    {

    }
}
```

Listato 4.6: Classe che rappresenta il Model rispecchiando la struttura di Croquet

La classe in questione è utilizzata per replicare le variabili condivise presenti nel Model del croquet-proxy, è necessario definire all'interno di questa classe le variabili che si vuole condividere con l'infrastruttura di Croquet. Ogni qualvolta vengano inviate publish con dei dati, l'intera classe viene serializzata in JSON e inviata assieme all'evento, o in caso di ricezione, i dati ricevuti da parte di Croquet vengono utilizzati per la ricostruzione di tale classe.

4.4 ClientView

```
public class ClientView : View
{
    private ClientModel data;
    public ClientView(ClientModel data) : base(data)
    {
        this.data = data;
        //Subscription
    }
}
```

Listato 4.7: Classe che rappresenta la View rispecchiando la struttura di Croquet

La classe corrente va a definire quella che è la View relativa al client, rispecchiando le funzionalità della classe View di Croquet.

Allinterno di essa possono essere effettuate le subscription agli eventi di croquet da intercettare, vengono anche definiti i metodi da eseguire in caso di ricezione della publish dedicata, in caso possono essere inseriti dei metodi dedicati all'invio di publish.

Capitolo 5

Realizzazione dell'applicazione

In questo capitolo verrà mostrata la tecnica di integrazione del caso di studio in un applicativo di Mixed Reality, attraverso due esempi esplicativi verrà dimostrato il funzionamento e la corretta configurazione e implementazione del client e del croquet-proxy.

5.1 CounterClient

In questo primo esempio viene data una dimostrazione base del funzionamento dell'infrastruttura realizzata, l'esempio consiste nel realizzare un contatore condiviso attraverso Croquet, il contatore in questione ogni secondo invierà un evento che comunica il proprio valore, ogni client collegato e che abbia effettuato la subscription a tale evento, emetterà una publish con lo scopo di aumentare il valore del contatore condiviso di un'unità.

5.1.1 Configurazione del croquet-proxy

Naturalmente per la corretta condivisione della variabile è necessario configurare il Model relativo a Croquet, così da definire la variabile condivisa e gli eventi alla quale effettuare le publish e i subscribe.

```
// my-model.js
import { Model } from 'croquet-proxy';
export class MyModel extends Model {
    init(options) {
        super.init(options);
        // Define your model data always in the 'data' property
        this.data = { counter: 0}
        this.subscribe('counter', 'update', this.setData);
        this.future(1000).tick()
```

```
    }

    tick() {
        this.publish('counter', 'updated', this.data);
        this.future(1000).tick()
    }

    setData(data) {
        this.data.counter++;
    }
}
```

Listato 5.1: Model del croquet-proxy relativo all'esempio che si vule implementare

All'interno del Model viene quindi realizzata la variabile condivisa all'interno di data, e viene effettuata la subscription all'evento che gestirà l'avanzamento del valore, ogni secondo verrà effettuato il publish del valore corrente.

5.1.2 Configurazione del client

Una volta definita la variabile condivisa che deve essere gestita, è opportuno configurare adeguatamente le classi ClientView e ClientModel per la corretta gestione di dati ed eventi e allacciare gli oggetti tridimensionali allo script attraverso la classe Main.

ClientModel

```
[System.Serializable]
public class ClientModel : ModelData
{
    public int counter;

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

    public ClientModel(int counter)
    {
        this.counter = counter;
    }
```

```
}
```

Listato 5.2: Model del Client di Mixed Reality relativo all'esempio che si vule implementare

La variabile condivisa gestita da Croquet come definito in precedenza deve essere replicata all'interno di questa classe per garantire il corretto funzionamento.

ClientView

```
public class ClientView : View
{
    private ClientModel data;
    public ClientView(ClientModel data) : base(data)
    {

        base.Subscribe("counter", "updated", (data) =>
            OnCounterUpdated(data));
        this.data = data;
    }

    private void OnCounterUpdated(System.Object tmp)
    {
        ClientModel tmpC = (ClientModel)tmp;
        this.data.counter = tmpC.counter;
        base.Publish("counter", "update", 1);
    }
}
```

Listato 5.3: View del Client di Mixed Reality relativo all'esempio che si vuole implementare

All interno della classe corrente, come descritto nei capitoli precedenti vengono gestiti gli eventi di subscribe e publish relativi al client, si può notare la subscription all evento generato da Croquet e la publish che viene effettuata ogni volta che si riceve l'aggiornamento del valore.

MainCounter

Classe principale del Client, che permette l'aggancio degli oggetti grafici alle variabili condivise.

```
public class MainCounter : MonoBehaviour
{
    private ClientModel data = new ClientModel(0);
    private ClientView view;
    private GameObject debugGameObj;
    private TextMeshProUGUI debugConsole;

    // Start is called before the first frame update
    void Start()
    {
        this.view = new ClientView(this.data);
        debugGameObj = GameObject.Find("DebugTxt");
        debugConsole = debugGameObj.GetComponent<TextMeshProUGUI>();
    }

    // Update is called once per frame
    void Update()
    {
        this.debugConsole.text = this.data.counter.ToString();
    }
}
```

Listato 5.4: Main script del Client di Mixed Reality relativo all'esempio che si vuole implementare

5.2 CubeSharing

Una volta dimostrato il corretto funzionamento attraverso l'esempio definito in precedenza, ritengo opportuno mostrare un ulteriore esempio che vada a simulare il reale scopo del progetto, ovvero, dare la possibilità di gestire un cubo tridimensionale condiviso.

Visto l'attuale gestione delle componenti condivise è necessario implementare due Client differenti, uno che svolga il ruolo di "Master", cioè colui che ha la possibilità di muovere il cubo, mentre il resto dei Client condivisi saranno implementati come "Slave", avendo la possibilità di vedere il Cubo muoversi nell'ambiente circostante, riproducendo una sorta di ambiente condiviso di presentazione.

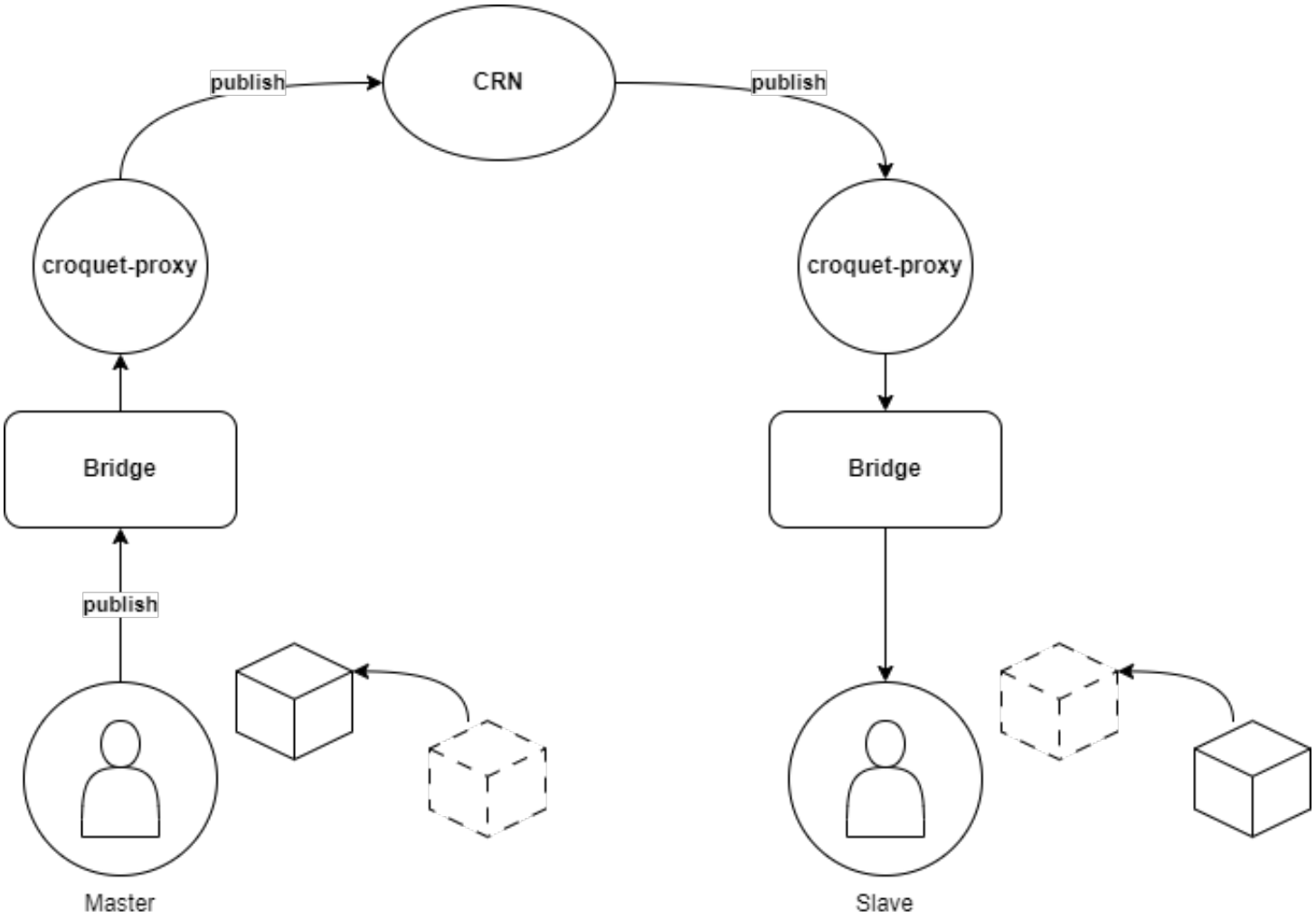

Figura 5.1: Struttura logica del funzionamento

5.2.1 Configurazione del croquet-proxy

Come nell'esempio precedente è prima di tutto necessario occuparsi della configurazione del Model di Croquet, visto che l'esempio si incentra sul condividere la posizione di un cubo nello spazio corrente, il set di variabili condivise sarà quindi composto da tre variabili ovvero le tre coordinate cartesiane.

```
import { Model } from 'croquet-proxy';
export class MyModel extends Model {
    init(options) {

        super.init(options);
```

```
        // Define your model data always in the 'data' property
        this.data = { cubeX : 0, cubeY : 0, cubeZ : 0}
        this.subscribe('cube', 'sendData', this.setData);

    }

    setData(data) {

        this.data.cubeX = data.cubeX;
        this.data.cubeY = data.cubeY;
        this.data.cubeZ = data.cubeZ;
        this.publish('cube', 'updated', this.data);

    }
}
```

Listato 5.5: Model di Croquet relativo all'esempio che si vuole implementare

Analizzando il codice è essenziale descrivere la strategia utilizzata per la modalità di condivisione che si vuole avere, viene effettuata la Subscription all'evento emesso dal Master e ogni qualvolta si riceva l'aggiornamento dei valori viene effettuato il Publish per notificre gli Slave.

5.2.2 Configurazione del Master

Verranno successivamente introdotte le tre classi fondamentali per definire la struttura del client che svolge il ruolo di Master.

MasterModel

La classe corrente andrà quindi, come nell'esempio precedente, a replicare le variabili condivise da gestire.

```
[System.Serializable]
public class MasterModel : ModelData
{
    public float cubeX;
    public float cubeY;
    public float cubeZ;

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

```
    public MasterModel(float x, float y, float z)
    {
        this.cubeX = x;
        this.cubeY = y;
        this.cubeZ = z;
    }
}
```

Listato 5.6: Model del client Master relativo all'esempio che si vuole implementare

MasterView

La View relativa al Master svolgerà quindi solo il ruolo di effettuare publish con il valore corrente delle coordinate cartesiane del Cubo.

```
public class MasterView : View
{
    private MasterModel data;

    public MasterView(MasterModel data) : base(data)
    {
        this.data = data;
    }

    public void SendData(MasterModel newData)
    {
        base.Publish("cube", "sendData", newData);
    }
}
```

Listato 5.7: View del client Master relativo all'esempio che si vuole implementare

MasterMain

Al fine di inviare i dati costantemente per catturare ogni movimento del cubo, l'invio dell'evento di publish è stato inserito all'interno del metodo Update, ciò comporta l'invio dei dati a ogni frame dell'applicazione.

```
public class MasterMain : MonoBehaviour
{
    private MasterModel data;
    private MasterView view;
    private GameObject testCube;
    // Start is called before the first frame update
    void Start()
    {
        testCube = GameObject.Find("TestCube");
        this.data = new MasterModel(testCube.transform.position.x,
            testCube.transform.position.y,
            testCube.transform.position.z);
        this.view = new MasterView(this.data);
    }

    // Update is called once per frame
    void Update()
    {
        this.view.SendData(new
            MasterModel(testCube.transform.position.x,
            testCube.transform.position.y,
            testCube.transform.position.z));
    }
}
```

Listato 5.8: Main del client Master relativo all'esempio che si vuole implementare

5.2.3 Configurazione dello Slave

Introdotto il codice del Client Master è necessario ora definire il codice relativo alle tre classi fondamentali dello Slave.

SlaveModel

La classe corrente definisce il Model dello Slave, come è possibile immaginare è identico al Model del Master e al Model di croquet.

```
[System.Serializable]
public class SlaveModel : ModelData
{

    public float cubeX;
    public float cubeY;
    public float cubeZ;

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

    public SlaveModel(float x, float y, float z)
    {
        this.cubeX = x;
        this.cubeY = y;
        this.cubeZ = z;
    }
}
```

Listato 5.9: Model del client Slave relativo all'esempio che si vuole implementare

SlaveView

La View implementata per la parte Slave effettua la subscription all'evento di update e aggiorna i dati correnti.

```
public class SlaveView : View
{
    private SlaveModel data;
    public SlaveView(SlaveModel data) : base(data)
    {
        base.Subscribe("cube", "updated", (data) =>
            OnCubeUpdated(data));
        this.data = data;
    }

    private void OnCubeUpdated(System.Object tmp)
    {
        SlaveModel tmpC = (SlaveModel)tmp;
        this.data.cubeX = tmpC.cubeX;
```

```
        this.data.cubeY = tmpC.cubeY;
        this.data.cubeZ = tmpC.cubeZ;
    }
}
```

Listato 5.10: View del client Slave relativo all'esempio che si vuole implementare

SlaveMain

La classe Main dello Slave gestisce infine l'effettivo aggiornamento della posizione, ridefinendo la posizione dell'oggetto tridimensionale all'interno del metodo Update.

```
public class SlaveMain : MonoBehaviour
{
    private SlaveModel data;
    private SlaveView view;
    private GameObject testCube;

    // Start is called before the first frame update
    void Start()
    {

        testCube = GameObject.Find("TestCube");
        this.data = new SlaveModel(testCube.transform.position.x,
            testCube.transform.position.y,
            testCube.transform.position.z);
        this.view = new SlaveView(this.data);

    }

    // Update is called once per frame
    void Update()
    {
        testCube.transform.position = new Vector3(data.cubeX,
            data.cubeY, data.cubeZ);
    }
}
```

Listato 5.11: Main del client Slave relativo all'esempio che si vuole implementare

Capitolo 6

Analisi dei risultati

Definiti tutti i punti che compongono il caso di studio e mostrati gli esempi di implementazione, è necessario addentrarsi in un'analisi dei risultati ottenuti.

Effettivamente il risultati ottenuti dimostrano la possibilità di implementare applicazioni di Mixed Reality condivisa attraverso l'infrastruttura descritta. Naturalmente lo sviluppo utilizzando tale struttura comporta alcune limitazioni che porterebbero a rallentare o portare problematiche in caso di progetti su larga scala.

6.1 Problematiche

La struttura definita per la gestione di applicativi di Mixed Reality condivisa può risultare poco efficace in caso di progetti che comprendono molteplici oggetti tridimensionali da condividere, ciò comporterebbe l'invio di grandi quantità di dati e a frequenze elevate portando il sistema sotto stress. Naturalmente si tratta di una supposizione derivante dal fatto che il client e il bridge al momento sono stati implementati al solo scopo di dimostrare il funzionamento, e non tenendo conto delle questioni di ottimizzazione ed efficienza.

Una seconda problematica degna di nota è la possibilità di avere dei delay dovuti alla connessione di rete, trattandosi di tre applicativi separati che necessitano un collegamento via rete per cominciare tra loro, è necessario eseguire le tre parti in modo che ci siano meno interferenze possibili. Per lo svolgimento dei test ad esempio sono stati eseguiti il croquet-proxy e il bridge all'interno dello stesso computer collegando poi il visore via wireless.

6.1.1 Ownership

Un aspetto completamente assente all'interno del progetto è il concetto di ownership degli oggetti tridimensionali, ovvero l'implementazione di un mec-

canismo che permetta di identificare chi sta interagendo con un determinato oggetto. Visto che non è stata ancora implementata questa funzionalità non si è in grado di utilizzare lo stesso software client nel caso di sessioni condivise, ma è necessario suddividere i client in Master e Slave, l'implementazione di tale meccanismo è essenziale nel caso si debbano gestire stanze collaborative.

6.2 Lavori futuri

Per quanto riguarda l'aspetto di lavori futuri, sicuramente è consigliabile svolgere dei lavori di ottimizzazione e analisi delle prestazioni, cercando di effettuare il porting di tutta l'infrastruttura all'interno del visore Hololens 2, in modo da garantire la minore latenza possibile.

Un aspetto interessante da analizzare e sviluppare sicuramente è la realizzazione di una struttura base per applicazioni di grandi dimensioni, implementato il concetto di ownership, allargando quindi il set di funzionalità e controlli forniti di default dall'applicativo in modo da rendere lo sviluppo il più generico e semplice possibile.

Infine ritengo necessario effettuare un'esplorazione accurata del Croquet Microverse, già effettuata in passato, ma senza riuscire a ottenere risultati notevoli. L'integrazione di Mixed Reality condivisa a un mondo di Metaverso, come definito in precedenza, permetterebbe di gestire una qualsiasi casistica della condivisione in modo più semplice e intuitivo, garantendo oltretutto di realizzare quindi una collaborazione perfetta tra i partecipanti.

Conclusioni

Si può ritenere di aver raggiunto l'obiettivo di questo caso di studio, andando a definire la possibilità effettiva di poter realizzare applicazioni di Mixed Reality condivisa tramite l'utilizzo di Croquet. Nonostante il progetto possa risultare non ottimizzato, è una vera e propria dimostrazione dell'effettiva potenzialità che la piattaforma Croquet ha da offrire e come possano essere sfruttate anche in casistiche ostiche e specifiche. L'effettivo raggiungimento dell'obiettivo potrebbe portare a definire quindi una nuova tecnica di condivisione che permetta lo sviluppo di applicazioni per la realtà mista condivisa, con la possibilità di garantire il completo controllo e la totale personalizzazione di tutti gli elementi che compongono la scena, garantendo quindi la libertà di implementazione.

Ringraziamenti

Vorrei, prima di tutto, ringraziare i miei genitori e mia sorella, che hanno avuto un ruolo fondamentale per me nel raggiungimento di questo traguardo, sostenendomi e spronandomi nei momenti più difficili.

Un particolare ringraziamento va al professor Alessandro Ricci e al dottor Samuele Burattini, che mi hanno seguito costantemente nello sviluppo di questa tesi, facendomi appassionare alle tematiche trattate.

Vorrei ringraziare tutti gli amici che mi hanno accompagnato in questo corso di studio, conosciuti in aula, con la quale ho condiviso esperienze uniche. Ultimo, ma non per importanza, un ringraziamento speciale va alla mia seconda famiglia, il B0RSA TEAM, e a coloro che reputo come fratelli: Alessandro A., Alessandro P., Edoardo, Giacomo, Pietro e Sebastian. Grazie per tutte le esperienze passate insieme e grazie per il vostro sostegno.

Bibliografia

[1] Che cos'è realtà mista toolkit 2? "https://learn.microsoft.com/it-it/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05".

[2] Che cos'è un ologramma? "https://learn.microsoft.com/it-it/windows/mixed-reality/discover/hologram".

[3] Croquet 1.0.5. "https://croquet.io/docs/croquet/".

[4] The croquet reflector network (crn). "https://croquet.io/global-reflector-network/".

[5] Croquet worldcore 1.3.0 overview. "https://croquet.io/docs/worldcore/".

[6] Esperienze condivise nella realtà mista. "https://learn.microsoft.com/it-it/windows/mixed-reality/design/shared-experiences-in-mixed-reality".

[7] Hololens 2. "https://www.microsoft.com/it-it/hololens/hardware".

[8] Holoportation™. "https://www.microsoft.com/en-us/research/project/holoportation-3/".

[9] Introducing microsoft mesh. "https://www.microsoft.com/en-us/mesh".

[10] Introduction to node.js. "https://nodejs.dev/en/learn/".

[11] Mapping spaziale. "https://learn.microsoft.com/it-it/windows/mixed-reality/design/spatial-mapping".

[12] Model. "https://croquet.io/docs/croquet/Model.html".

[13] Openxr. "https://www.khronos.org/openxr/".

[14] Panoramica di microsoft mesh (anteprima). "https://learn.microsoft.com/it-it/mesh/overview?source=recommendations".

[15] Panoramica di rendering olografico. "https://learn.microsoft.com/it-it/windows/mixed-reality/develop/advanced-concepts/rendering-overview".

[16] Projectionworks mixed reality for manufacturing. "https://deltasigmacompany.com/mixed-reality/".

[17] Tutorial. "https://croquet.io/docs/microverse/tutorial-Tutorial.html".

[18] View. "https://croquet.io/docs/croquet/View.html".

[19] What is augmented reality or ar? "https://dynamics.microsoft.com/en-us/mixed-reality/guides/what-is-augmented-reality-ar/".

[20] What is mixed reality? "https://learn.microsoft.com/it-it/windows/mixed-reality/discover/mixed-reality".

[21] What socket.io is. "https://socket.io/docs/v4/".

[22] Mazzoli Alessandro. Sviluppo di applicazioni real-time multi-utente: un middleware basato sulla piattaforma croque. 2022. "https://amslaurea.unibo.it/26845/1/un_middleware_basato_sulla_piattaforma_Croquet.pdf".

[23] Shreya Mattoo. What is virtual reality? types, benefits, and real-world examples. September 2022. "https://www.g2.com/articles/virtual-reality".