ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

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

UN FRAMEWORK PER LA REALIZZAZIONE DI APPLICAZIONI DI EXTENDED REALITY COLLABORATIVE

Elaborato in
PERVASIVE COMPUTING

Relatore
Prof. ALESSANDRO RICCI

Corelatore
Dott. Ing. SAMUELE BURATTINI

Presentata da
ANNA VITALI

Anno Accademico 2022 – 2023

"Il coraggio non è la mancanza di paura, piuttosto la consapevolezza che qualcosa sia più importante della paura stessa"

Mia Thermopolis (Anne Hathaway), Il padre in una lettera.
dal film "Pretty Princess" di Garry Marshall

Indice

Introduzione

Douglas C. Engelbart nelle prime pagine dell'articolo "*Augmenting Human Intellect: a Conceptual Framework*" [6], descrive come l'intelletto umano possa essere "aumentato" grazie agli strumenti sempre più sofisticati che è possibile utilizzare, prendendo in considerazione il computer come il più promettente fra gli strumenti moderni, per poter riuscire a determinare soluzioni a possibili problemi del mondo, mostrando come esempio quella che viene definita la "*Mother of all demos*" (oN-Line System) [7], che nel 1968 illustrò per la prima volta un sistema di videoconferenza con condivisione dello schermo, in un *editor* di testo collaborativo in tempo reale.

I computer di oggi, tuttavia, non sono più fissi sulla nostra scrivania, ma grazie all'avvento delle tecnologie *mobile* e *wearable* abbiamo la possibilità di portarli con noi ovunque andiamo e anche di poterli indossare; un'altra evoluzione che si sta sempre di più espandendo, riguardante i nostri sistemi computerizzati, è relativa ai contenuti che questi offrono; essi, infatti, stanno iniziando ad uscire sempre di più dallo schermo e ad entrare nel nostro spazio fisico, i contenuti che vengono mostrati presentano, pertanto, una propria locazione nello spazio, che ci consente di poterli osservare da diversi punti di vista.

Oggi, ci troviamo nell'era dello *Spatial Computing*, quello che viene definito essere: il quarto paradigma del *Personal Computing*. Questo paradigma, viene supportato dalle tecnologie mobile e wearable soprattutto quelle che consentono all'utente di poter immergersi in un mondo aumentato, come gli *smart-glasses* e i visori, attraverso sistemi di *Augmented, Mixed* e *Virtual Reality* [3].

Con la promessa del Metaverso la corsa da parte dei fornitori di servizi e di soluzioni informatiche, a sviluppare sia applicazioni pensate per questi dispositivi, che strumenti stessi che consentono di accedervi, è aumentata sempre di più. Le principali aziende leader nel settore informatico, come Microsoft e Apple, stanno lavorando per poter lanciare nuovi e migliori hardware di *Mixed Reality* al fine di riuscire ad ottenere e garantirsi una quota di questo mercato in crescita.

Se l'obiettivo che questi sistemi hanno è quello di riuscire ad aumentare il

nostro mondo, per favorire la ricerca di soluzioni sempre più complesse e sempre più idonee a problemi che dobbiamo affrontare, un aspetto di cui sicuramente occorre tenere conto, è la possibilità di creare delle applicazioni, che consentano di condividere delle esperienze, permettendo agli utenti di poter collaborare fra loro. Infatti, se ci pensiamo, quante volte nella nostra vita ci è capitato di non riuscire a risolvere un problema da soli, ma con l'aiuto di qualcun altro siamo riusciti a raggiungere il nostro obiettivo? Allora, per poter trovare soluzioni ai nostri problemi, oltre che ad utilizzare i nuovi strumenti che sono messi a disposizione, anche la collaborazione è un aspetto importante da considerare.

Tra le possibili applicazioni in Mixed Reality è ampiamente considerato che i sistemi collaborativi rappresentino delle "*killer applications*" [8], vale a dire dei programmi software, con un'interfaccia utente percepita come abbastanza innovativa, da influenzare le tendenze informatiche e le vendite [35].

Tuttavia, riuscire a costruire applicazioni di eXtended Reality collaborative oggi, che coinvolgano diverse realtà e consentano a più utenti di accedervi, non è ancora semplice; questo dovuto sopratutto al problema della frammentazione del mercato, in cui sostanzialmente, le imprese che producono i loro dispositivi, per ragioni economiche, richiedono lo sviluppo di applicazioni apposite per i loro sistemi, che però non risultano essere compatibili con altri dispositivi offerti da altri fornitori. Se in più, all'interno della nostra applicazione, oltre che a considerare la portabilità che questa deve avere nei confronti dei diversi apparati, vogliamo considerare anche di supportare i differenti tipi di realtà, andando a parlare di interoperabilità fra i diversi sistemi, il compito diventa ancora più arduo.

Questa tesi, di conseguenza, ha come obiettivi quelli di, inizialmente, analizzare il concetto di eXtended Reality e che cosa significhi realizzare oggi applicazioni collaborative di questo tipo, mettendo in luce quali siano le diverse sfide che uno sviluppatore può trovarsi ad affrontare. Successivamente, si prevede di entrare nel merito della realizzazione di un *framewrok* in grado di mettere a disposizione del programmatore, un ambiente di sviluppo e un insieme di elementi chiave, che possano essere utilizzati per realizzare sistemi di questo tipo; affrontando le sfide determinate precedentemente, anche analizzando parte del panorama tecnologico presente e specifici strumenti, che possono essere adottati per fornire e supportare determinate funzionalità, necessarie per questo tipo di applicazioni.

Capitolo 1

eXtended Reality

All'interno di questo capitolo andremo a definire che cosa intendiamo parlando di *eXtended Reality*, partendo da un concetto più ampio a cui questi si relaziona lo *Spatial Computing*, il quale introduce un elemento fondamentale per questi sistemi, vale a dire la spazialità, cioè il mantenimento di un riferimento a uno spazio, utilizzato per mostrare i contenuti in una certa locazione, anche in riferimento a un determinato ambiente reale. Successivamente, nel capitolo, si andrà ad analizzare il *Reality Virtuality Continuum*, chiarendo le differenze fra le diverse realtà, che ricadono all'interno dell'eXtended Reality: *Augmented, Mixed* e *Virtual*.

1.1 Spatial Computing

Lo Spatial Computing è considerato essere il quarto paradigma del *Personal Computing* [3] successivo ai precedenti tre paradigmi di: *Personal Computing*, *Graphical Interface and Thinking* e *Mobile* [3].

Il primo paradigma, Personal Computing, riguardava l'avvento del primo *personal computer*, il quale presentava, per poter gestire l'interazione con l'utente, un'interfaccia di tipo testuale, successivamente, il secondo paradigma, rappresentato dal Graphical Interface and Thinking, ha introdotto un'evoluzione nelle interfacce utilizzate attraverso l'adozione di elementi grafici e colori, infine, il terzo paradigma è rappresentato dal Mobile, il quale ha segnato l'avvento della nascita e della diffusione dei dispositivi mobili, che hanno portato i computer ad essere alla portata delle nostre mani. Il quarto paradigma, Spatial Computing, farà sì che i computer inizino ad uscire dal piccolo schermo che fino ad ora li ha contenuti e giungano nel nostro spazio fisico.

Per capire che cosa si intende per Spatial Computing, prendiamo a riferimento la definizione data da Simon Greenwold nell'articolo "*Spatial Computing*" [12]:

> "I define spatial computing as human interaction with a machine in which the machine retains and manipulates referents to real objects and spaces.[...] Spatial computing proposes hybrid real/virtual computation that erodes the barriers between the physical and the ideal worlds.[...] Wherever possible the machine in space and space in the machine should be allowed to bleed into each other. Sometimes this means bringing space into the computer, sometime this means injecting computation into objects. Mostly it means designing systems that push through the traditional boundaries of screen and keyboard without getting hung up there and melting into interface or meek simulation."

Greenwold, sostanzialmente, definisce lo Spatial Computing come l'interazione con una macchina in grado di manipolare riferimenti di oggetti e spazi reali, capace di proporre una computazione ibrida reale/virtuale in grado di erodere i confini fra mondo fisico e un mondo ideale (definizione molto simile al concetto di Mixed Reality, "*the merging of real and virtual worlds*", introdotto da Paul Milgram [28]). Ad esempio, un sistema che consenta agli utenti di installare delle forme virtuali nello spazio reale che lo circonda, è un sistema di Spatial Computing.

Potrebbe venire da confondere lo Spatial Computing con campi affini quali, la modellazione 3D e il design digitale, tuttavia, esso si differenzia da questi ambiti, in quanto richiede che le forme e gli spazi di cui si occupa siano **preesistenti** e abbiano una valenza **reale**. Inoltre, non è detto che lo Spatial Computing presenti per forza uno spazio all'utente [12], quello che è importante è che nel sistema di computazione ci sia una rappresentazione interna dello spazio, la quale può anche essere implicita nei dati collezionati e la sua interazione con l'utente non deve necessariamente essere visiva o spaziale e non necessariamente deve essere tridimensionale. Ad esempio, un sistema che ci consenta di individuare punti salienti di una mappa, in riferimento a un ambiente reale, può essere qualificato come sistema di Spatial Computing [34].

Nei sistemi di eXtended Reality, tuttavia, lo spazio con cui ci troviamo ad avere a che fare è tridimensionale e i sistemi adottati avranno una rappresentazione di questo spazio in termini di **coordinate cartesiane**. I contenuti mostrati agli utenti, quindi, avranno una propria locazione all'interno di questa rappresentazione interna dello spazio, che il nostro sistema è in grado di mantenere, il che ci fornisce la capacità di collocare gli elementi del mondo aumentato in posizioni specifiche, anche in relazione allo spazio fisico reale in cui ci troviamo.

Possiamo notare come vi possa essere una relazione stretta fra la rappresentazione dello spazio mantenuta dal sistema e il luogo reale in cui accediamo alla nostra applicazione, tant'è che Shashi Shekar, Steven k. Feiner e Walid

G. Aref, nel loro articolo "*Spatial Computing*" [34], fanno riferimento ad esso come ad un insieme di elementi in grado di creare una nuova comprensione dei luoghi così come li conosciamo, di come visualizziamo la nostra relazione rispetto a questi e di come ci muoviamo all'interno di essi.

I moderni sistemi di eXtended Reality, come abbiamo accennato, sono in grado di incorporare gli elementi dello Spatial Computing riuscendo appunto a creare un nuova comprensione dello spazio, consentendoci di inserire elementi aumentati all'interno di esso, quali gli ologrammi, con cui l'utente sarà in grado di interagire, anche dandogli modo di muoversi all'interno di questo spazio, potendo osservare i contenuti da diversi punti di vista. L'utente, pertanto, nelle nostre applicazioni, avrà una propria locazione nella rappresentazione dello spazio mantenuta dal sistema, la quale determinerà la sua visione sul mondo.

1.2 Cosa intendiamo quando parliamo di eXtended Reality

Il termine eXtended Reality viene in genere adottato per indicare un ombrello al disotto del quale sono coperti i diversi tipi di realtà: Augmented, Mixed e Virtual [9].

Il nome eXtended Reality viene in genere riportato con l'acronimo XR, dove la "X" in questo caso non vuole indicare solamente il termine *extended*, ma può essere interpretato come una variabile per tutte quelle tecnologie presenti e future, che potranno ricadere all'interno del suo spettro (figura 1.1).

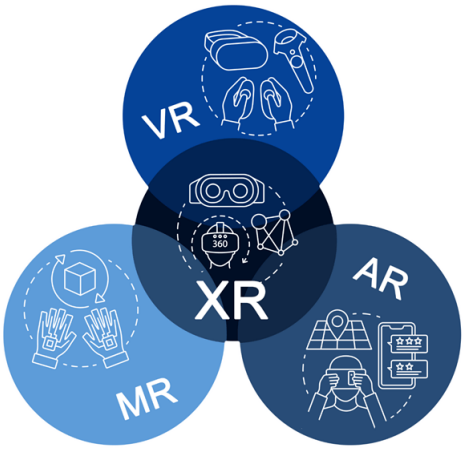


Figura 1.1: eXtended Reality Spectrum

Le moderne tecnologie XR con cui è possibile lavorare oggi hanno la possibilità di acquisire informazioni dell'ambiente in cui l'utente si trova e creare delle esperienze spaziali uniche, anche dal punto di vista dell'interazione con gli elementi del mondo, ad esempio, possono consentire agli utenti di poter creare una mappa 3D in real-time, della stanza in cui si trovano, attraverso il loro smartphone.

Tuttavia, l'eXtended Reality non riguarda solamente il 3D o il foto-realismo, le interfacce immersive che possono essere presentate attraverso questi sistemi, infatti, offrono esperienze di visualizzazione e interazione che possono cambiare il nostro senso dello spazio, andando ad estendere (da qui il termine "*extended*") la nostra esperienza vissuta nello spazio reale [9].

1.3 Il Reality Virtuality Continuum

Nel definire il termine eXtended Reality abbiamo detto che esso ci raffigura uno spettro al disotto del quale possiamo far ricadere i concetti di: Augmented, Mixed e Virtual Reality. Ora, cercheremo di capire quali siano le caratteristiche che queste realtà presentano e che ci consentono di poterle andare a distinguere, dandoci anche la possibilità di meglio comprendere le loro proprietà, analizzando quello che viene definito il *Reality Virtuality Continuum*.

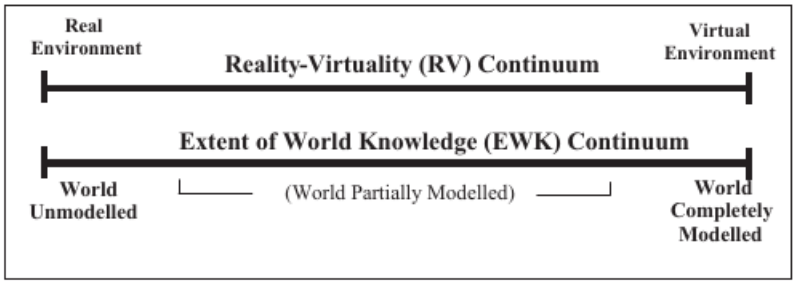


Figura 1.2: Reality Virtuality Continuum

La figura 1.2 ci mostra una rappresentazione del *Reality Virtuality Continuum*, che possiamo utilizzare per poter effettuare una prima distinzione fra quello che consideriamo **reale** e quello che, invece, può essere considerato **virtuale**.

Gli ambienti puramente reali (*Real Environment*) e gli ambienti virtuali (*Virtual Environment*) esistano certamente come entità separate, non devono essere considerati semplicemente come alternative tra loro, ma piuttosto come poli che si trovano alle estremità opposte del *Reality Virtuality Continum*. La posizione di qualsiasi ambiente, o mondo, lungo questo continuum coincide con la sua posizione lungo un continuum parallelo indicato come *Extent of*

World Knowledge (EWK). In particolare, quest'ultimo termine, ci consente di fare riferiamo alla portata della conoscenza presente all'interno del computer sul mondo presentato. Come indicato nella figura, all'estremità destra dell'RV continuum vi sono gli ambienti virtuali, i quali devono essere necessariamente completamente modellati per poter essere renderizzati. All'estremo sinistro, invece, consideriamo gli ambienti reali come rappresentazioni di un mondo, o regione, completamente non modellata [28].

In base a dove le nostre realtà si collocano sulla linea dell'RV continuum, possiamo capire a quale "modello" del mondo esse facciano riferimento.

1.3.1 Posizionamento delle diverse reltà nel Reality Virtuality Continuum

Le differenti realtà prima citate, possono collocarsi all'interno del *Reality Virtuality Cntinuum* come raffigurato nella sottostante figura 1.3.

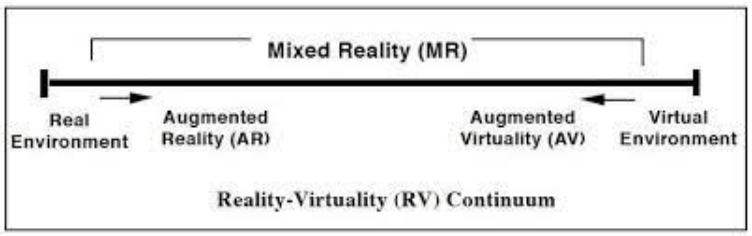


Figura 1.3: Posizione delle diverse realtà considerando il Reality Virtuality Continuum

Come possiamo vedere, l'Augmented Reality è posizionata nella metà di destra, in riferimento agli ambienti reali, in quanto in grado d**i arricchire la nostra percezione del mondo fisico** riuscendo a sovrapporre media allineati spazialmente in tempo reale; ad esempio, può alterare la visione dell'ambiente da parte dell'utente aggiungendo grafica computerizzata per trasmettere informazioni passate, presenti o future su un luogo o un oggetto [34].

La Virtual Reality, invece, viene posta nella metà di destra del *Reality Virtuality Continum*, in quanto, si tratta di un'avanzata *human-computer-interface*, in grado di simulare un ambiente realistico (mondo virtuale), in cui il partecipante all'esperienza può muoversi, vedere da diverse angolazioni, raggiungere punti all'interno di esso, afferrare e rimodellare elementi al suo interno [41], senza però avere alcuna visione del mondo reale, infatti, in questo tipo di applicazioni, la visione dell'utente del mondo reale viene oscurata. L'obiettivo principale della Virtual Reality è quello di posizionare il partecipante in un

ambiente virtuale, in grado di darli la sensazione di "essere lì" [41] , sebbene il mondo mostrato sia puramente sintetico.

Per quanto riguarda la Mixed Reality, il modo più semplice per visualizzare un ambiente di realtà mista è quello in cui gli oggetti del mondo reale e del mondo virtuale vengono presentati insieme all'interno di un unico display, quindi ovunque negli estremi del *Reality Virtuality Contimum* [40]. In questo tipo di applicazioni, l'utente ha la possibilità di vedere il mondo reale e ha percezione del fatto che si trova in questo mondo, non più in uno virtuale, pertanto si aspetta anche che l'esperienza che sarà in grado di provare sia la più reale possibile; egli avrà la capacità di muoversi nel mondo fisico e nel mondo virtuale allo stesso modo e simultaneamente, per poter consentire questo, la Mixed reality, si occupa di integrare fra loro funzionalità offerte sia da sistemi di Augmented che di Virtual Reality alle quali è in grado di aggiungere capacità quali: *environmental understanding*, *human understanding*, audio spaziale e gestione di posizioni e del posizionamento in spazi fisici e virtuali.

1.3.2 Differenze fra le diverse realtà

Augmented Reality Sebbene sia l'Augmented che la Mixed Reality consentano all'utente di avere una visione del mondo reale, il tipo di esperienza offerto da queste due tecnologie è differente. Nel primo caso il sistema adottato **non effettua**, o **non è in grado di realizzare**, un mapping spaziale dell'ambiente in cui l'utente si trova, la sua **comprensione dell'ambiente può essere limitata**, ad esempio, al semplice riconoscimento delle superfici piane presenti all'interno della stanza, in modo da riuscire a posizionare un ologramma sul pavimento, però oltre a questo non ha conoscenza delle caratteristiche dello spazio reale in cui l'utente è in grado di muoversi, non riconosce la posizione dei mobili della stanza o la loro forma.

Mixed Reality Nella Mixed Reality, invece, i sistemi possiedono una **consapevolezza spaziale**, che li rende in grado di comprendere e analizzare le caratteristiche del mondo reale che l'utente è in grado di vedere e in cui l'utente è in grado di muoversi. Nella Mixed Reality, gli ologrammi sono visti dall'utente come se fossero degli oggetti reali, quindi ad esempio, se interagendo con essi uno di questi finisse dietro un mobile o sotto una scrivania, egli non sarebbe più visibile all'utente almeno che questi non decida di spostare il suo punto di vista.

Questa capacità, che gli ologrammi hanno in sistemi di realtà mista, che li consente di reagire alle caratteristiche del mondo, è data da una proprietà fondamentale chiamata **occlusione**, la quale non è presente in sistemi di Augmented Reality. L'occlusione, pertanto, è quella proprietà che ci consente di

marcare maggiormente la distinzione fra Augmented e Mixed Reality, rendendo gli ologrammi capaci di reagire ai cambiamenti e alle proprietà del mondo reale; per poterla ottenere non bastano semplicemente sensori appositi che ci consentono di percepire il mondo, ma occorre adottare anche algoritmi di intelligenza artificiale e semantica, che consentono al dispositivo non solo di analizzare i dati raccolti, ma anche di attribuirli un significato.

Virtual Reality La differenza sostanziale che ci consente, invece, di discernere la Virtual Reality rispetto agli altri due tipi di realtà, è la visione e modellazione del mondo che questi sistemi sono in grado di fornire ed effettuare, come detto precedentemente, nella realtà virtuale il **mondo mostrato all'utente è interamente costruito** dal programmatore ed egli non ha la possibilità di poter vedere il mondo reale, il quale li viene opportunamente nascosto tramite l'**oscurazione della vista**. Il mondo virtuale che viene mostrato al partecipante, non deve per forza riportare le caratteristiche del mondo reale in cui questi si trova, ma può essere modellato a piacimento in base all'applicazione che si vuole realizzare e alle caratteristiche che si vuole che questo abbia.

(a) Augmented Reality (b) Mixed Reality (c) Virtual Reality

Figura 1.4: Distinzione fra le differenti realtà

In conclusione, alla luce di quello che abbiamo detto, possiamo osservare come la distinzione fra realtà mista e aumentata sia più sottile, rispetto a quella che possiamo osservare fra queste due tipologie di realtà nei confronti della Virtual Reality. Difatti, si potrebbe dire che l'unica cosa che Augmented e Mixed Reality hanno in comune con la Virtual Reality è solo il termine realtà.

Capitolo 2

Applicazioni collaborative

Per poter definire che cosa significa realizzare applicazioni di eXtended Reality collaborative partiamo dal concetto di collaborazione.

Collaborare significa partecipare insieme con altri a un lavoro a una produzione [38]; l'opera di chi collabora è il risultato di tale lavoro.

Stando a questa definizione, la collaborazione presuppone sempre che, nel lavoro che viene svolto sia sempre coinvolto un **gruppo di individui**, quindi, non possiamo parlare di collaborazione quando abbiamo a che fare con un utente **isolato**. La collaborazione rompe l'isolamento e per raggiungere lo scopo finale del nostro lavoro ci impone di interagire con altre persone.

Quando l'attività collaborativa svolta dagli individui è supportata da sistemi computerizzati, in informatica, si parla di Computed Supported Cooperative Work (CSCW) [13], termine formalmente adottato nel 1984 da Grudin per indicare lo sviluppo di sistemi informatici, che danno la possibilità di supportare il lavoro di piccoli gruppi di persone.

Come si può vedere dalla figura 2.1, Grudin proponeva un modello concettuale costituito da più cerchi concentrici, in cui ogni anello mostra un obiettivo dello sviluppo di sistemi informatici e il cliente principale o l'utente della tecnologia risultante. Per quanto riguarda i CSCW, il sistema di riferimento è costituito da reti di computer e comunicazioni mediate da computer o workstation e il cliente di riferimento è rappresentato da piccoli gruppi di persone.

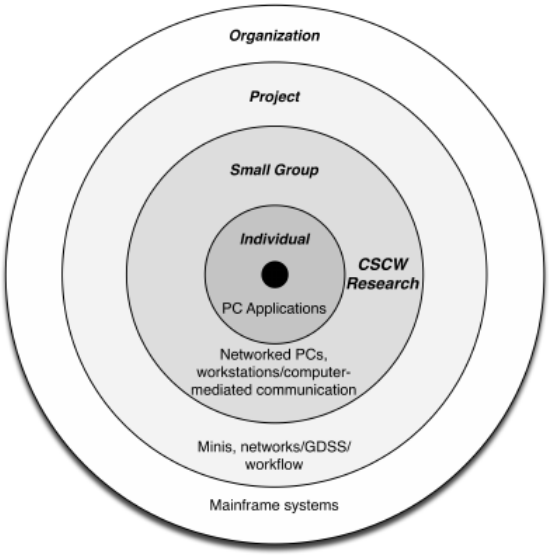


Figura 2.1: Modello per lo spazio di progettazione di CSCW

2.1 Le dimensioni della collaborazione

I sistemi CSCW si focalizzano sul lavoro cooperativo e sono diretti a individuare le strategie migliori che possono essere messe in campo per poterlo consentire. Tuttavia, di sistemi cooperativi non ne esiste uno solo, ma diversi e il metodo tradizionale utilizzato per distinguerli prevede l'utilizzo della matrice spazio-tempo di Johansesn [18] (in figura 2.2).

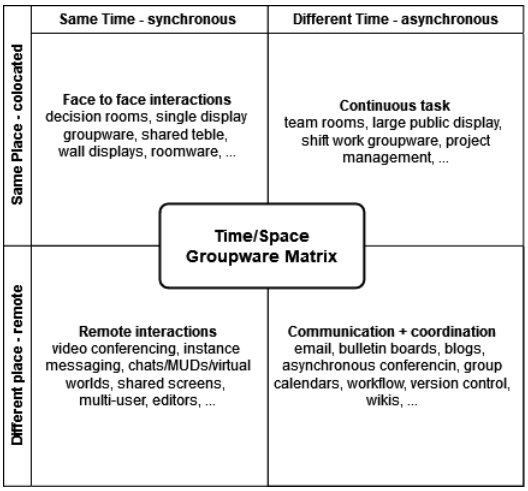


Figura 2.2: Matrice spazio-tempo

Come possiamo vedere dalla figura 2.2, la matrice è costituita da due dimensioni, una del **tempo** e una dello **spazio**. L'asse del tempo è diviso in **sincrono** e **asincrono**, quello dello spazio, invece, in **co-locato** e **remoto**.

Quando si pensa ad esperienze condivise, tipicamente viene da riferirsi a esperienze di tipo sincrono, in cui tutti insieme lavoriamo e cooperiamo per il progetto allo stesso momento [25], tuttavia, possiamo avere anche esperienze condivise di tipo asincrono dove, ad esempio, un messaggio o una nota vengono lasciati per la persona successiva che continuerà il lavoro e anche in questo tipo di scenari si può comunque instaurare un regime di collaborazione. Inoltre, non è detto che l'esperienza che si viene a creare sia per forza sincrona o asincrona, ma potrebbe essere entrambe, il che significa ad esempio, che inizialmente l'esperienza può essere asincrona e tutti lavoriamo insieme, fino a quando un membro del gruppo non decide di uscire e allora, si sceglie di lasciare in sospeso il lavoro fatto (salvandone lo stato) per poi riprenderlo successivamente, dando luogo a un'esperienza asincrona.

Per quanto riguarda lo spazio, invece, quando un'esperienza condivisa ha luogo nella stessa località essa si dice co-locata, i partecipanti si trovano nello stesso luogo e comunicano fra loro direttamente, quando invece, le persone si trovano in luoghi diversi l'esperienza si dice essere remota, essi partecipano alla collaborazione stando in luoghi differenti connettendosi da remoto [25] [8]. Anche in questo caso, non è detto che l'esperienza che si venga a creare escluda l'uno o l'altro scenario, è possibile avere casi di esperienza ibrida in cui, ad esempio, alcuni partecipanti si trovano nella stessa stanza ed altri invece si trovano in luoghi diversi.

Queste due dimensioni, che per anni sono state considerate la norma per poter distinguere il tipo di lavoro cooperativo di riferimento, che si voleva andare a realizzare tramite i sistemi CSCW, **non sono** tuttavia sufficienti a considerare tutti i diversi aspetti che possono emergere nel lavoro di collaborazione [21].

La critica maggiore che viene riportata alla matrice spazio-tempo di Johansen, per la classificazione dei sistemi CSCW, è data da CP. Lee e D. Pane nell'articolo "*From The Matrix to a Model of Coordinated Action (MoCA)*" [21] in cui si dice che, innanzitutto, l'utilizzo della matrice per individuare gli scenari di lavoro cooperativo era inadeguato fin dal principio, in quanto questa matrice fu inizialmente creata avendo come riferimento solamente i *meeting* elettronici ed era presentata assieme ad un'altra che analizzava la differenza fra piccoli e grandi gruppi di lavoro, considerandoli come scenari concettualmente distinti.

Come detto prima, infatti, i sistemi CSCW, inizialmente, si focalizzavano su piccoli gruppi di lavoro, dove per piccolo possiamo intendere un gruppo composto da meno di sette persone [25]; tuttavia, quando consideriamo lavori

ed esperienze collaborative il numero di individui coinvolti può essere molto ampio, in alcuni casi si può anche arrivare a parlare di partecipazione di massa.

Per questi motivi, ritenendo insufficiente la matrice spazio-tempo per poter decomporre i sistemi cooperativi, CP. Lee e D. Pane propongono di ampliare le dimensioni con cui poter andare a classificare i CSCW passando da due a sette. Le quali verranno illustrate nei successivi paragrafi.

Synchronicity Questa dimensione fa riferimento a quanto visto prima per la matrice di Johansen in merito all'asse temporale, a partire da un estremo di essa si ha un lavoro completamente sincrono, mentre dall'altro estremo si ha un lavoro completamente asincrono.

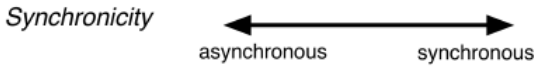


Figura 2.3: Synchornicity

Physical distribution Anche questa dimensione è stata presa dalla precedente matrice e riguarda il luogo in cui le azioni avvengono, in un estremo queste si verificano nella stessa località geografica, nell'altro estremo, invece, queste si verificano in località geografiche differenti.

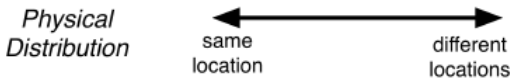


Figura 2.4: Physical Distribution

Scale Questa dimensione fa riferimento al numero di partecipanti coinvolti nella collaborazione. In genere, gli sviluppatori tendono a pensare secondariamente al numero di partecipanti supportati dall'esperienza, quando invece, dovrebbe essere uno dei primi aspetti da considerare, questo in quanto ricerche sulla teoria del coordinamento [22][21] ci dicono che, quando più attori (per noi inteso i partecipanti all'esperienza) perseguono insieme degli obiettivi, devono fare cose per organizzarsi che un singolo attore non dovrebbe fare. Quindi, detto più semplicemente, lavorare insieme ad altri per raggiungere un obiettivo, **richiede un'organizzazione diversa del lavoro, rispetto a quella che si avrebbe lavorando in singolo** e la scala per questo motivo deve essere tenuta in considerazione.

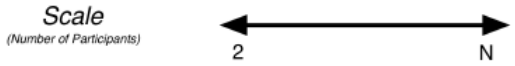


Figura 2.5: Scale

Number of Community of Practice La terza dimensione, come dice il nome, fa riferimento al "numero di comunità di pratica", gli estremi di questa dimensione vanno da zero Comunity of Practice fino a N. Che cos'è però una *CoP*? Per poter capire il concetto che sta dietro a questo termine, si può fare riferimento al significato della parola **interdisciplinarità** nei gruppi di lavoro [21].

Un gruppo di lavoro, difatti, può essere costituito da collaboratori di diverse discipline e con differente formazione. Molto spesso, quando si lavora a un progetto, per incentivare la creatività e l'innovazione persone con differenti capacità vengono messe insieme; inizialmente, la loro collaborazione può risultare difficile, specialmente se provengono da realtà molto diverse fra loro e la forza del gruppo sta proprio nel riuscire a trovare una coesione e nello sviluppo di un gergo unico, che possa favorire il lavoro da svolgere.

La dimensione della NCoP si riferisce, pertanto, alla diversità culturale fra i membri del gruppo, che può comprendere diversità nelle norme, nelle pratiche negli strumenti, nel linguaggio etc. che vengono utilizzati. Meno è elevato il NCoP più coeso sarà il gruppo, più è elevato questo numero più differenze vi saranno fra i diversi membri del team. Per questa dimensione si va verso un'estremità in cui vi sono piccoli gruppi di lavoro omogenei, composti da persone con background e formazione simili, all'altro estremo in cui vi sono persone che presentano background e formazione molto diverse fra loro, che non appartengono ad alcuna CoP.

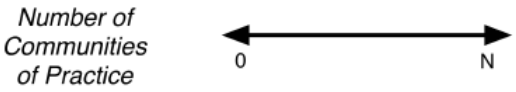


Figura 2.6: Number of Community of Practice

Nascence questa dimensione che in italiano si traduce con "nascita", non fa riferimento a un concetto di novità quanto più a un tipo speciale di "instabilità ricco di potenzialità future" [21]. Che cosa significa questo? Per capire questa dimensione, all'interno dell'articolo, di CP. Lee e D. Pane, si fa riferimento alla nozione di *Boundary Negotating Artifact* [20], vale a dire ad artefatti temporanei e non standardizzati, che possono venire creati per supportare la collaborazione.

Il tipo di collaborazione che stiamo andando a creare con il nostro sistema, infatti, può richiedere l'adozione o la creazione di artefatti non standardizzati o non esistenti in precedenza, che abilitino meccanismi necessari per l'attività che vogliamo andare a svolgere o che rappresentino il risultato dell'attività collaborativa svolta. Quindi, la dimensione della nascita fa riferimento a qualcosa di nuovo messo in campo per la collaborazione o dovuto alla collaborazione.

La dimensione della Nascence viene misurata sulla base del tipo di lavoro svolto, se questo rappresenta un lavoro di routine, allora l'aspetto della nascita descritto prima sarà basso se invece esso rappresenta un lavoro di sviluppo, tale aspetto potrà essere più elevato.

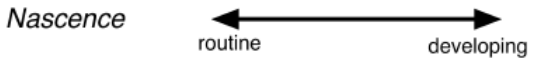


Figura 2.7: Nascence

Planned Permanence Dimensione che fa riferimento alla "permanenza pianificata" dell'accordo di collaborazione. In particolare, la Planned Permanence fa riferimento a un tipo di permanenza pianificata o presunta [21], in quanto, molto spesso, non è possibile sapere a priori quanto durerà un'azione coordinata, vale a dire un'azione svolta da uno o più soggetti nel team di lavoro nel regime di collaborazione, ne è semplice decidere a che punto qualcosa sia effettivamente permanente. In poche parole, con questa dimensione si analizza se l'azione compiuta nell'attività di collaborazione sia a breve o lungo termine, in entrambi i casi i partecipanti all'esperienza devono creare pratiche, artefatti e termini condivisi, quindi formulare un gergo comune, per poter raggiungere i propri scopi.

Tramite questa dimensione vogliamo analizzare se la collaborazione che si viene a creare sia **temporanea**, non destinata a durare nel tempo, oppure **permanente**, destinata a durare, ad esempio, la collaborazione che si viene a creare fra un gruppo di individui per montare una tenda da campeggio per il week-end si intende temporanea, mentre quella che si viene a creare per costruire una casa in cui viverci nel lungo periodo, rappresenta una collaborazione con un grado di permanenza più elevato. Notare comunque, che una collaborazione temporanea non è per forza più semplice di una permanente [21].

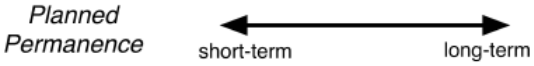


Figura 2.8: Planned Permanence

Turnover L'ultima dimensione si riferisce alla stabilità della composizione dei partecipanti, vale a dire la rapidità con cui i partecipanti possono uscire e entrare nel lavoro. Per questa dimensione si guarda se la collaborazione è chiusa e privata, quindi, limitata a un certo gruppo e/o numero di persone, oppure se sia aperta, dove per aperta possiamo intendere anche aperta a chiunque.

Di conseguenza, considerando questi due estremi di scenari di collaborazione, aperti e chiusi, nel primo caso il turnover potrà essere più elevato e veloce, gli individui potranno entrare e uscire quando vogliono e con la frequenza che desiderano, mentre nell'altro il turnover sarà più basso e lento, in quanto gli individui partecipanti saranno di meno e probabilmente, entreranno o usciranno dalla collaborazione con minor frequenza rispetto a prima.

Figura 2.9: Turnover

Tramite l'aggiunta delle dimensioni di: *Scale*, *Number of Community of Practice*, *Planned Permanence* e *Turnover*, siamo in grado di avere un quadro abbastanza ampio, almeno più ampio rispetto a prima (considerando solo le dimensioni di spazio e tempo), sui tipi di collaborazione che si possono instaurare.

Adesso scendiamo maggiormente in dettaglio per quello che riguarda esperienze collaborative di eXtended Reality, analizzando quali sono i possibili scenari che, tenendo conto di queste dimensioni e altre considerazioni, possono essere individuati e che meglio ci possono aiutare nel progettare il nostro sistema CSCW.

2.2 Analisi dello scenario di riferimento per esperienze di eXtended Reality

Quando facciamo riferimento esperienze condivise e collaborative di eXtended Reality, alludiamo a sistemi costituiti da più persone in grado di partecipare a un'esperienza, attraverso dispositivi pensati per differenti realtà, che siano in grado di fornire ai partecipanti la visione di un certo contenuto aumentato, dando la possibilità di interagire con esso e fra loro.

Gli scenari possibili di esperienze che si possono andare a creare, sono differenti, anche in relazione all'analisi delle dimensioni, definite precedentemente, che può essere fatta.

Per facilitare l'individuazione dello scenario di riferimento, Microsoft, nella sua piattaforma di e-learning[1], propone allo sviluppatore che voglia realizzare esperienze di Mixed Reality collaborative, di provare a rispondere a sei semplici domande [25], le quali possano aiutarlo a comprendere meglio che tipo di espereinza si vuole andare a realizzare. Queste domande possono essere generalizzate non solo per lo sviluppo di applicazioni Mixed Reality, ma anche per le altre realtà.

How are they sharing? Questa prima domanda vuole fare ragionare lo sviluppatore su come avviene la condivisone dei contenuti e a seconda della risposta data è possibile individuate tre categorie principali di esperienze condivise [25]: **presentazione**, in cui un utente si occupa di condividere il contenuto che gli altri sono in grado di vedere ed egli è l'unico responsabile per questo, quindi, solo lui ha la possibilità di modificarlo, **collaborazione**, in cui le persone lavorano insieme per raggiungere un obiettivo comune e **guida**, quando una persona aiuta un'altra a risolvere un determinato problema, fornendo assistenza tramite un rapporto 1:1.

Rispondere a questa domanda può farci ragionare sulla dimensione Nascence vista prima, in relazione al tipo di attività che viene svolta e informazioni che vengono scambiate e anche sulla dimensione Number of Community of Practice, a seconda degli individui che partecipano all'esperienza e dalle capacità che essi hanno di fornire un contributo, sulla base della loro formazione ed esperienza.

What is the group size? La seconda domanda vuole far riflettere lo sviluppatore sulla dimensione della scala definita precedentemente, in particolare la dimensione del gruppo si considera piccola se il numero di partecipanti è inferiore a sette, grande se è maggiore di sette [25]. Inoltre, nel rispondere a questa domanda anche la dimensione del Turnover viene considerata, in quanto la dimensione del mio gruppo può aumentare o diminuire dinamicamente a seconda di questo aspetto.

Where is everyone? Questa domanda vuole, invece, far ragionare sulla dimensione della spazialità. La risposta che viene data ad essa influenza: come le persone comunicano, ad esempio, se è previsto l'utilizzo di avatar, per poter comunicare in remoto con gli altri partecipanti e quali oggetti vedono, se questi sono tutti condivisi e se vi è la necessità di adattare l'ambiente fisico utilizzato [25].

[1] `https://learn.microsoft.com/en-us/`

When are they sharing? La quarta domanda ci impone di ragionare sulla dimensione temporale vista prima. La risposta a tale domanda può influenzare: la persistenza degli oggetti e dell'ambiente, quindi, richiede anche un'analisi della dimensione della permanenza pianificata, l'utilizzo della prospettiva, per esempio ricordandosi qual'era il punto di vista dell'utente sulla scena quando egli ha deciso di lasciarla etc.

How similar are their physical environments? Questa domanda fa riferimento alla dimensione spaziale, se abbiamo un'esperienza di tipo co-locata, infatti, gli ambienti dei partecipanti saranno identici, se invece abbiamo un'esperienza remota questi è più probabile che siano diversi, tuttavia, possono presentare comunque caratteristiche simili, ad esempio, le sale conferenze presentano in genere lo stesso mobilio: un grande tavolo al centro contornato da diverse sedie.

Rispondere a questa domanda può influenzare: come le persone interagiscono con gli oggetti, ad esempio, se l'esperienza richiede di visualizzare un modello a dimensione naturale per poter discutere i diversi aspetti della sua realizzazione, i partecipanti devono avere la possibilità di potersi muovere nello spazio potendo girare attorno al modello analizzandone le sue caratteristiche, la scala degli oggetti e come questi vengono presentati [25].

What devices are they using? Quest'ultima domanda, per chi vuole realizzare esperienze che coinvolgano diverse realtà è molto importante, in quanto prende in considerazione un aspetto rilevante che è l'**asimmetria** dei dispositivi adottati. Come detto precedentemente, infatti, i diversi dispositivi pensati per le diverse realtà presentano differenze nell'esperienza che l'utente è in grado di provare, tali differenze però devono essere opportunamente gestite, in modo da dare a tutti la possibilità di accedere e provare l'esperienza.

Rispondere a questa domanda, per le dimensioni viste prima, ci fa ragionare sia in termini di Nascence che di Number of Community of Practice. La dimensione della Nascence può essere considerata, in relazione ai modi che possono essere messi in campo per poter consentire l'interoperabilità fra i diversi sistemi e ragioniamo anche in termini di Number of Community of Practice, in relazione alle funzionalità che i diversi sistemi possono avere, i quali danno la possibilità agli utenti partecipanti di avere determinate capacità.

2.2.1 Alcuni scenari possibili

Giunti fino a qui abbiamo chiaro quali sono gli aspetti che possiamo considerare per individuare il tipo di applicazione che vogliamo realizzare, tuttavia, realizzare applicazioni collaborative che coinvolgano differenti realtà o facciano

riferimento a una realtà specifica, non è una novità, diversi sono gli esempi che possono essere trovati in letteratura e nell'articolo "*Revisiting Collaboration through Mixed Reality: The Evolution of Groupware*" [8] è stata fatta un'analisi di 100 applicazioni di Mixed Reality collaborative, al fine di poter definire una tassonomia dei sistemi che possono essere realizzati e analizzarne lo sviluppo nel tempo.

Nell'articolo gli scenari che sono stati individuati, analizzando i 100 lavori presi a riferimento, sono fondamentalmente cinque, i quali vengono di seguito riportati:

1. **Remote expert**, lavori in cui vi è una persona esperta che guida da remoto una persona in locale nello svolgimento di un particolare task;
2. **Shared workspace**, punto fermo per lavori che includono una forte attenzione su uno spazio di lavoro fisico e virtuale combinato;
3. **Shared experience**, scenario che include lavori che si concentrano sull'esperienza collaborativa offerta ai partecipanti/collaboratori piuttosto che sul particolare progetto su cui stanno lavorando;
4. **Telepresence**, include lavori maggiormente concentrati sulla comunicazione in remoto fra due o più partecipanti;
5. **Co-annotation**, riguarda sistemi che inseriscono annotazioni virtuali su un oggetto o ambiente di interesse che possano essere letti dagli altri.

L'applicazione di eXtended Reality che andremo a realizzare potrà ricadere su **uno o più di questi scenari**, a seconda dell'analisi che andremo a svolgere, rispondendo alle domande poste precedentemente e alle sette dimensioni individuate, che consentono di discernere diversi aspetti e caratteristiche del nostro sistema.

2.3 Sfide da affrontare per la collaborazione nell'eXtended Reality

Precedentemente abbiamo detto che collaborare significa lavorare insieme, non in solitario, di conseguenza, quando si vuole realizzare sistemi per supportare la collaborazione stiamo cercando un modo per modellare la socialità [8].

La collaborazione, per poter avvenire, richiede di avere uno spazio di lavoro comune. Questo concetto di "*common field of work*" è stato proposto e analizzato da Schmidt e Simon [32], i quali notarono che uno spazio di lavoro comune

è il luogo in cui avviene quello che viene definito "lavoro coordinativo". Tale lavoro è costituito dall'interdipendenza di più attori che, nello svolgimento delle loro attività individuali, cambiando il loro campo di lavoro, **cambiano anche quello degli altri** e gli attori interagiscono fra loro tramite il cambiamento dello stato del campo di lavoro comune.

Questa definizione, di lavoro coordinativo, ci interessa non soltanto perché fa riferimento alla collaborazione, ma perché nei sistemi di eXtended Reality possiamo avere una **chiara rappresentazione** di questo spazio, rappresentato dal contenuto aumentato condiviso.

Tuttavia, realizzare esperienze di questo tipo è tutt'altro che semplice, ricordiamo, infatti, che dal momento in cui coinvolgiamo più utenti, ciascuno con un proprio dispositivo, abbiamo a che fare con un sistema distribuito e le problematiche che questo comporta. Inoltre, queste applicazioni, per loro natura, richiedono di considerare degli aspetti che in altri sistemi non necessariamente si hanno: la vista della **prospettiva degli altri utenti**, come avviene la **condivisione degli elementi olografici** e lo **spazio di riferimento** adottato.

Andiamo adesso ad analizzare questi aspetti, mettendo in luce quali sono le sfide che uno sviluppatore, che si addentri nella realizzazione di questo tipo di sistemi, si trova ad affrontare.

2.3.1 Il punto di vista dell'altro

Le prime ricerche sui sistemi CSCW si concentrarono sulla comprensione dei comportamenti collaborativi negli spazi fisici, da questi studi, emerse che due elementi teorici richiedano un adeguata esplorazione. In primo luogo, capire come abilitare la consapevolezza per i collaboratori di **conoscere chi si trova nello spazio di lavoro** e di che cosa questi stia facendo. In secondo luogo, articolare una comprensione di **come le informazioni visive possano supportare la collaborazione** [8].

Le persone che partecipano a un'esperienza condivisa e collaborativa in un'ambiente virtuale, si aspettano di avere lo stesso grado di consapevolezza dello spazio di lavoro e dell'altro, che si ha nel mondo fisico. In spazi co-locati, nella vita reale, questa conoscenza ci viene fornita da tre aspetti [14][8], che possono essere riportati anche nel mondo virtuale:

1. Linguaggio del corpo delle persone;

2. Conversazione e gesti espliciti;

3. Artefatti dello spazio di lavoro.

In un ambiente virtuale in cui abbiamo la **perdita del nostro corpo fisico**, ricercatori hanno studiato come l'utilizzo di avatar [16], ad esempio, possa favorire la conoscenza delle attività promosse dagli altri e la loro consapevolezza, dando la possibilità anche di prevedere azioni future. Per capire quanto questo sia importane, L. D. Segal, nell'articolo "*effects of checklist interface on non verbal crew communication*" [33], descrive come i piloti spendano più della metà del loro tempo nel guardare le attività del loro co-pilota, per poter coordinare le loro attività sulla base delle azioni compiute dall'altro.

La rappresentazione dell'altro nel nostro spazio di lavoro comune, ci dà la possibilità di osservare e meglio comprendere il suo punto di vista, ad esempio, in un'applicazione di condivisione e modifica dei documenti, il cursore dell'altro ci dà l'informazione di dove il suo focus sia in questo momento. Allo stesso modo, nel realizzare le applicazioni collaborative per le differenti realtà dobbiamo tenere conto, non essendo da soli, di quello che l'altro sta facendo ed osservando e per riuscirci possiamo sfruttare le capacità che i dispositivi stessi ci offrono. Ad esempio, ormai molti visori moderni, come HoloLens 2 danno la possibilità di tracciare lo sguardo dell'utente, funzionalità che possiamo usare per capire dove l'altro sta guardando o ancora i *controllers* che possono essere usati per fornire l'input dell'utente in un ambiente di Virtual Reality, ci possono fornire informazioni su dove le sue mani si trovino nello spazio.

Quando la collaborazione che si vuole creare ha come argomento centrale un oggetto visibile nel mondo virtuale e la sua analisi, capire il punto di vista dell'altro e i movimenti/gesti che questi produce per supportare la sua analisi è molto importante. Tuttavia, in ambienti 3D anche solo mostrare l'azione di puntamento verso un'oggetto di interesse può risultare complicato [8]. Le informazioni visive condivise forniscono un importante risorsa di conversazione per la collaborazione, ma nel nostro tipo di applicazioni, le viste che gli individui hanno sull'ambiente **possono cambiare**, non sono fisse, di conseguenza, occorre affrontare le sfide per la comunicazione date da prospettive di vista disgiunte.

Il contenuto mostrato agli utenti ha una propria locazione nello spazio fisico, tramite i nostri sistemi vogliamo consentire loro di muoversi liberamente e in modo indipendente dagli altri nello spazio, per poterlo osservare con diverse prospettive; tuttavia, vogliamo consentire anche ai collaboratori di comprendere il punto di vista degli altri sull'ambiente; occorre, pertanto, formulare strategie che ci consentano di congiungere questi due aspetti.

2.3.2 Spazio di riferimento condiviso

Nel nostro caso, tutte le applicazioni che vengono create che siano di Augmented, Mixed o Virtual Reality, prevedono l'utilizzo di un sistema di rife-

rimento, il quale viene utilizzato per posizionare gli oggetti nel mondo visto dall'utente. Di conseguenza, dato che le diverse applicazioni che si vengono a creare immergono l'utente in un nuovo spazio, in cui è possibile vedere il contenuto aumentato, per l'obiettivo che vogliamo raggiungere, come già abbiamo analizzato precedentemente, è **necessario avere uno spazio di lavoro comune**, rappresentato nel nostro caso dal mondo aumentato.

Le applicazioni di nostro interesse generano quello che viene chiamato un "*3D collaborative virtual environment*", vale a dire, un ambiente in cui tutti i partecipanti osservano lo stesso spazio 3D e sono in grado di vedere gli oggetti di questo spazio con la stessa dimensione, posizione relativa e orientazione [2].

Gli utenti che partecipano all'esperienza presentano uno stesso sistema di riferimento, cioè un sistema Cartesiano tramite cui posizioni e orientazioni possono essere misurate [2]; bisogna però fare attenzione a questo punto, in quanto, possiamo distinguere due tipologie diverse di spazio di riferimento comune:

- **Assoluto**, in cui gli utenti sono effettivamente in grado di vedere l'ologramma nello stesso punto del loro spazio fisico. Lo spazio di riferimento del sistema, in questo caso, può essere ottenuto attraverso l'utilizzo di ancore spaziali, le quali rappresentano un sistema di coordinate fisso che può essere tracciato da un sistema operativo [15], oppure tramite *marker* posizionati all'interno dell'ambiente, che vengono adottati come punti di riferimento dal sistema. Essi, infatti, rappresentano dei *patterns* che possono essere individuati e riconosciuti da algoritmi di *computer vision*, rendendoli in grado di determinare la loro locazione all'interno dello spazio [15];

- **Relativo**, in questo caso i partecipanti non vedono esattamente gli ologrammi nella stessa posizione dello spazio fisico, ognuno di essi ha il proprio personale punto di vista sul mondo aumentato [2], pertanto, ogni partecipante avrà un proprio spazio di riferimento e ad esempio, nel caso in cui l'ologramma si trovi in posizione `(1, 1, 1)` del mondo, tutti i partecipanti saranno in gradi di vederlo in questa stessa locazione, in relazione al proprio sistema di coordinate.

Un'osservazione che possiamo fare in merito alla distinzione precedente, fra spazio di riferimento comune assoluto e relativo, è che nel caso dell'assoluto, è **necessario un punto comune di riferimento**, per riuscire a calibrare correttamente il nostro sistema, altrimenti occorrerebbe fondere assieme i diversi spazi di riferimento dei diversi utenti, operazione molto complessa da eseguire.

Figura 2.10: Esempio di applicazione Microsoft Mesh

Adottare un punto di riferimento comune è, fino ad ora, la strategia più semplice che possiamo considerare per riuscire a vedere gli elementi del mondo aumentato nello stesso punto, ed è tra l'altro la strategia che è stata adottata da Microsoft per l'applicazione Mesh [24], in cui gli utenti, in un contesto di Mixed Reality, sono in grado di collaborare assieme, vedendo gli ologrammi nello stesso punto del mondo, grazie a un elemento comune per tutti presente all'interno della scena: un tavolino, che ci rappresenta un **ologramma ancorato a un punto dello spazio** e il cui sistema di coordinate viene adottato per posizionare tutti gli altri elementi del mondo, in modo tale che tutti i partecipanti siano in grado di vederli nella stessa posizione rispetto a questo tavolo (figura 2.10).

La dimensione della spazialità

Osservando la discussione fin ora intrattenuta, su uno spazio di riferimento comune, emerge che la spazialità dei nostri sistemi è una caratteristica importante e per poterla gestire in modo corretto è utile ragionare in termini di: **contenimento**, **topologia**, **distanza**, **orientazione** e **movimento** [2].

Tramite il contenimento i partecipanti si sentono parte di un certo spazio o mondo aumentato, possiamo fare riferimento a questo concetto anche attraverso il termine di **immersione**, gli utenti si sentono immersi all'interno di un certo ambiente e possono essere in grado di percepire la presenza degli altri. Tramite la topologia lo spazio è strutturato e i partecipanti sono in grado di ragionare in termini di **mutua località e distanza degli elementi**

presentati, avendo anche la capacità di muoversi in questo spazio e quando il contenuto presentato dai nostri sistemi ha una propria locazione nell'ambiente, riuscire a muoversi nello spazio, mantenendo un riferimento fisso di dove si trovi l'oggetto all'interno di questi, per poterlo osservare da diverse angolazioni, è una capacità molto importante. Infine, l'utilizzo di uno spazio di riferimento consistente ci dà la possibilità di percepire correttamente, l'orientazione, la direzione e il movimento degli elementi, rendendoci capaci di fare riferimento ad oggetti condivisi in modo adeguato.

Tuttavia, queste abilità di ragionare in termini di posizione reciproca, orientazione e di movimento all'interno di un ambiente comune vengono a un determinato **costo**, per questo gli sviluppatori devono attentamente considerare i requisiti per le loro applicazioni. Un requisito per la conoscenza dell'appartenenza al gruppo può essere soddisfatto dalla proprietà di contenimento; un requisito per la manipolazione di ologrammi condivisi richiede uno spazio di riferimento comune; l'esigenza di sostenere una popolazione ampia e dinamica di utenti può richiedere punti di vista navigabili individuali [15].

2.3.3 Gli ologrammi in uno spazio di lavoro condiviso

Gli elementi cardine dei nostri programmi di eXtended Reality sono gli ologrammi e in un mondo condiviso essi svolgono un ruolo importante per noi, in quanto, come entità facenti parte di questo mondo, anche loro saranno condivisi, nel senso che potranno essere visti dai diversi utenti.

Poiché abbiamo parlato di ologramma condiviso, verrebbe da pensare che di questi nelle nostre applicazioni **ve ne sia uno solo** per ogni utente e che tutti i partecipanti collegati vedano quell'unico elemento, tuttavia, questo **non è del tutto vero**. In un'applicazione di questo genere, infatti, gli utenti vedranno una copia dell'oggetto condiviso. Ognuno di loro possiede uno strumento differente per poter accedere all'applicazione, il quale si occupa di renderizzare il contenuto in modo da renderlo visibile, quindi, non avendo un solo dispositivo di *rendering* per tutti, ma avendo a che fare con un **sistema distribuito**, ogni utente vedrà una **copia del contenuto del mondo olografico** generata dal suo dispositivo.

Questo aspetto, introduce una problematica non da poco nella realizzazione delle nostre applicazioni, in quanto, una volta che il programma viene dispiegato non vi è alcuna copia originale dell'ologramma e se ogni utente vede una propria copia del contenuto visto degli altri, come facciamo ad ottenere una collaborazione fra questi e fare in modo che se uno di loro modifichi lo stato di un oggetto, tutti vedono il nuovo stato, come se di oggetto in realtà ce ne fosse uno solo? A questo punto, entra in gioco un altro aspetto chiave per la realizzazione di sistemi di questo tipo, che è la **sincronizzazione**.

Gli oggetti condivisi del mondo sono e devono essere sincronizzati fra loro, questo implica che se un utente effettua una modifica sulla sua copia di un ologramma, allora grazie alla sincronizzazione che si ha con le altre repliche, tutti gli utenti saranno in grado di vedere questa modifica e avere l'illusione che di questo ologramma ve ne sia uno solo.

Attenzione però, in quanto, sebbene in precedenza, siano stati distinti i casi di esperienza sincrona *vs* asincrona, quando parliamo di artefatti condivisi nel lavoro di gruppo, sia che si stia collaborando in questo momento o in momenti distinti, la sincronizzazione è importante per poter vedere lo stesso stato degli elementi del mondo condiviso. La sincronizzazione è, sostanzialmente, lo **strumento che utilizziamo per rendere invisibile il fatto che ogni utente possiede una propria copia del contenuto olografico**.

Un ologramma in un ambiente condiviso, per quanto detto, ci rappresenta un'entità replicata fra i diversi utenti partecipanti all'esperienza, con cui essi possono avere la possibilità di interagire, la quale possiede una propria posizione all'interno dello spazio e il cui stato è sincronizzato con le altre repliche, in modo da consentire che un suo cambiamento possa essere percepibile da tutti i partecipanti coinvolti nell'esperienza.

Condivisione e sincronizzazione un'osservazione importante

Per quello che abbiamo detto nelle precedenti sezioni, per poter collaborare in questo tipo di applicazioni occorre condividere ed essere sincronizzati.

Supponiamo, quindi, di realizzare un'applicazione che consenta di mostrare a due utenti un contenuto aumentato raffigurante un cubo, il quale se viene toccato modifica il suo stato cambiando di colore, scegliendone casualmente uno fra un set di colori possibili; i nostri partecipanti all'esperienza decidono di collaborare assieme per assegnare al cubo un colore appropriato, quindi, quando uno dei due tocca l'ologramma anche l'altro deve essere in grado di vedere il colore raggiunto dal cubo, in modo da poter dare la propria opinione sul risultato. Per poter fare questo l'utente che ha effettuato la modifica comunicherà all'altro il nuovo stato del cubo e grazie a questa condivisione di informazioni, i due ologrammi visti dai partecipanti saranno sincronizzati.

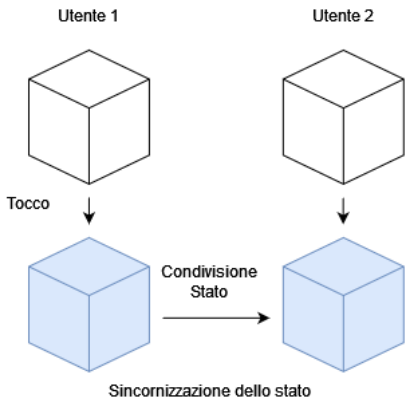


Figura 2.11: Schema riassuntivo esempio descritto

Un'importante osservazione che possiamo puntualizzare, analizzando l'esempio fatto, è che **per mezzo della condivisione delle informazioni**, fra i diversi partecipanti all'esperienza, che **siamo in grado di poter ottenere una sincronizzazione dello stato del mondo**, infatti, grazie al fatto che il primo utente condivide lo stato del cubo, a seguito della modifica effettuata, le due copie risultano essere sincronizzate.

In conclusione, possiamo pertanto dire, che la condivisione dello stato degli ologrammi e dei cambiamenti del mondo, è quella capacità che dobbiamo riuscire ad ottenere nei nostri sistemi, per poter riuscire a ottenere la sincronizzazione di questi, fra i diversi utenti e raggiungere l'obiettivo preposto. Tramite di essa siamo in grado di far percepire agli utenti la partecipazione degli altri e di realizzare l'esperienza condivisa che volevamo.

Capitolo 3

Tecnologie abilitanti per la collaborazione nell'eXtended Reality

Alla luce dell'analisi di che cosa significhi realizzare applicazioni di eXtended Reality collaborative, effettuata nel capitolo 2, possiamo già iniziare a definire alcuni requisiti di base che queste applicazioni devono avere:

- garantire il supporto a diversi dispositivi pensati per le diverse realtà;
- dare la possibilità agli utenti di vedere lo stesso mondo aumentato;
- consentire agli utenti di poter vedere le modifiche degli altri, effettuate ad oggetti condivisi.

Premettendo che questi siano i requisiti base che vogliamo andare a soddisfare, vediamo quali sono possibili strumenti innovativi, nati di recente, che possono essere utilizzati per poter raggiungere il nostro scopo.

In particolare, rispetto al passato, in cui per poter realizzare un'applicazione di questo tipo, era consigliato adottare appositi motori grafici, che consentissero di lavorare in uno spazio 3D, come ad esempio Unity [36], andando pertanto a realizzare soluzioni che richiedevano di essere installate sui dispositivi, per poter essere eseguite; oggi, è possibile sviluppare **soluzioni basate direttamente sul Web**, che non richiedono all'utente di installare alcuna applicazione, ma che danno la possibilità di poter accedere ai contenuti che si vuole offrire, direttamente collegandosi a un apposito sito.

Pertanto, le soluzioni che verranno illustrate in questo capitolo fanno riferimento a un contesto di sviluppo Web, tuttavia, alcune delle tecnologie illustrate possono essere comunque utilizzate attraverso motori grafici come

Unity o Unreal Engine [11], ma all'interno di questo capitolo non verranno trattati questi aspetti.

3.1 WebXR

WebXR [31] è l'*Application Program Interface (API)*, che ci consente di poter portare le nostre applicazioni di eXtended Reality sul Web. Più precisamente, WebXR, consiste in un insieme di standard utilizzati assieme per poter supportare il *rendering* di scene 3D su hardware progettato per presentare mondi virtuali (Virtual Reality) o per aggiungere immagini grafiche al mondo reale (Augmented Reality).

Per poter raggiungere questo obiettivo, le capacità chiave offerte da questa API sono di seguito elencate:

- individuazione di dispositivi di VR o AR compatibili;
- renderizzazione della scena 3D a un appropriato *frame-rate*;
- creazione di vettori rappresentanti il movimento dei controlli di input.

I dispositivi compatibili con WebXR includono i *fully-immersive 3D headsets* dotati di *motion* e *orientation tracking*, *smart-glasses*, che siano in grado di sovrapporre la grafica computerizzata al mondo reale e telefoni cellulari, in grado di aumentare la realtà catturando il mondo con una fotocamera, aggiungendo alla scena inquadrata immagini generate dal computer.

WebXR, per quanto detto, è un API che è autonomamente in grado di riuscire a **gestire le differenze** che i **diversi dispositivi**, utilizzati dagli utenti per accedere all'applicazione, possono presentare, dando la possibilità al programmatore di non doversi preoccupare di questo aspetto e di concentrarsi solo sui contenuti che si vuole andare ad offrire.

Quest'API è in grado di supportare sia sessioni di realtà aumentata che di realtà virtuale, utilizzando la medesima interfaccia, risultando essere compatibile anche con dispositivi di Mixed Reality, in quanto, in grado di supportare entrambi questi tipi di esperienza.

Questo strumento, quindi, ci torna molto utile per soddisfare il requisito precedente: **garantire il supporto a diversi dispositivi pensati per le diverse realtà**, offrendoci di fatto un API che ci consente di astrarre dal particolare *device* che verrà utilizzato dall'utente, per accedere all'esperienza, e dalle differenze che l'interfaccia di interazione può presentare, le quali verranno gestite automaticamente.

WebXR, tuttavia, come riportato dalla documentazione [29], **non è una tecnologia di rendering**, difatti, non fornisce alcuna funzionalità per poter

gestire i dati 3D o il rendering di questi sul display. Sebbene WebXR gestisca i tempi, la pianificazione e i vari punti di vista rilevanti quando si disegna la scena, non sa come caricare e gestire i modelli, né come renderizzarli, creare *texture* e così via. Questa parte è **interamente a carico del programmatore** e per poter fare questo, è possibile adottare altre librerie come WebGL [19] o altri frameworks (basati sempre su WebGL) pensati per rendere la gestione di tutti questi aspetti molto più semplice.

3.1.1 XR Session

L'elemento principale che WebXR richiede di andare a specificare, per mezzo della sua interfaccia, il quale li consente di capire quale tipo di sessione fra Augmented o Virtual Reality si voglia andare a supportare, è l'`XRSession` [29].

Sessioni di Virtual Reality

Ci sono due differenti modalità di sessione messi a disposizione per la Virtual Reality:

- `inline`, presenta la scena renderizzata all'interno del contesto di un documento nel browser web e non richiede hardware XR speciale per essere visualizzata. Con questo tipo di modalità è possibile visualizzare la scena 3D semplicemente aprendo la pagina Web con un proprio dispositivo, che non deve per forza essere un visore di Virtual Reality. Tuttavia, l'esperienza che si è in grado di vivere è fortemente basata su quella offerta dai dispositivi pensati per questo tipo di realtà, pertanto, non si sarà in grado di vedere il mondo esterno e si verrà posizionati all'interno di un mondo puramente sintetico [29];

- `immersive-vr`, questa modalità di sessione richiede un dispositivo XR, come un visore e prevede di sostituire l'intero mondo con la scena renderizzata utilizzando i display che è possibile avere su ciascuno degli occhi dell'utente.

Sessioni di Augmented Reality

Per le sessioni di Augmented Reality, invece, dato che questa realtà genera sempre un'esperienza immersiva, in cui la scena viene inserita nel mondo che circonda l'utente, è presente un'unica modalità, la quale viene indicata con il termine di `immersive-ar`.

3.2 Tecnologie di rendering

Precedentemente, abbiamo detto che per il rendering delle nostre applicazioni è possibile utilizzare la libreria WebGL [19], tuttavia, poiché la grafica 3D e la realtà mista in particolare, coinvolgono molta matematica, spesso complessa, per la gestione dei dati raccolti e lo svolgimento dei differenti compiti, risulta essere molto difficile utilizzare direttamente questa libreria, per questo motivo sono nati appositi frameworks, pensati per agevolare l'utilizzo di WebGL, in grado di fornire tutta una serie di vantaggi e di funzionalità che possiamo sfruttare per poter realizzare i nostri sistemi [30].

Di questi frameworks, quelli attualmente che risultano essere i più utilizzati sono: Babylon.js [1] e Three.js [37].

3.2.1 Babylon.js

Babylon.js ci rappresenta un motore grafico, basato su WebGL, che può essere utilizzato per la costruzione delle nostre applicazioni. Come riportato dalla *home-page* del sito ufficiale, l'obiettivo dei creatori di Babylon è quello di poter costruire uno dei più potenti, belli, semplici e *open* motori di rendering al mondo [1].

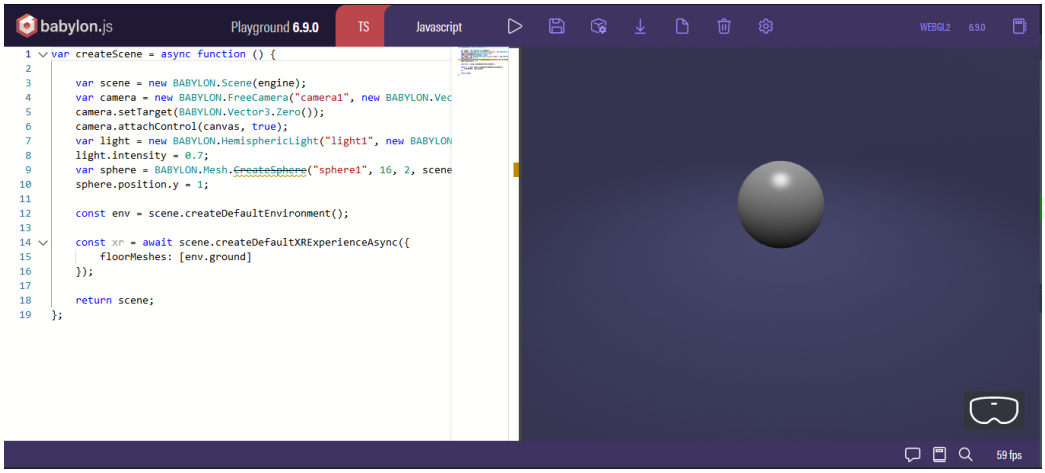


Figura 3.1: Babylon.js Playground

Per poter rendere più semplice lo sviluppo attraverso di esso, uno strumento messo a disposizione da Babylon, chiamato **Babylon Playground**, dà la possibilità agli utenti di poter scrivere e testare direttamente i risultati della scena

costruita, ad esempio nella figura 3.1 è possibile visionare una scena [1] costruita sfruttando il supporto a WebXR, che il framework mette a disposizione.

Gli esempi realizzati attraverso il *playground*, possono essere salvati dagli utenti e messi a disposizione degli altri attraverso la piattaforma, così che chiunque abbia la possibilità di visionarli e utilizzarli come riferimento.

Infine, un altro strumento molto utile fornito da Babylon.js è **Sandbox** (figura 3.2), il quale consente all'utente di visionare, gli elementi che possono essere inseriti all'interno della scena assieme alle loro caratteristiche, in un ambiente 3D, potendo modificare anche alcune delle loro proprietà direttamente agendo in questo ambiente.

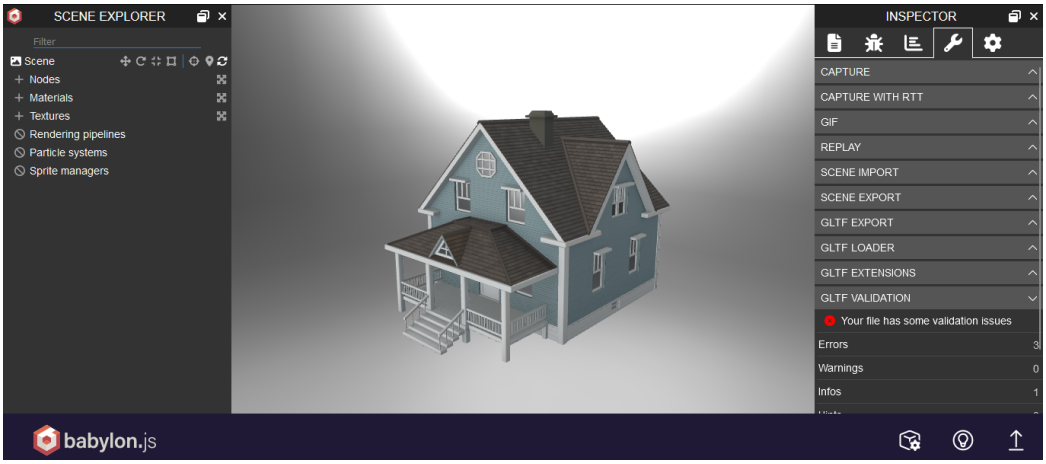


Figura 3.2: Babylon.js Sandbox

3.2.2 Three.js

Three.js è una libreria 3D pensata per rendere il più semplice possibile visualizzare contenuti 3D su una pagina Web [37].

Similmente a Babylon, Three.js, mette a disposizione sul proprio sito ufficiale un'insieme di esempi [2], che possono essere utilizzati dagli sviluppatori, per poter comprendere meglio come sia possibile inserire e utilizzate determinate funzionalità nell'applicazione che si intende andare a realizzare.

Anche questo framework si basa su WebGL e fornisce un supporto a WebXR, dandoci la possibilità di poter realizzare applicazioni di eXtended Reality.

La figura 3.3, mostra un esempio di applicazione WebXR realizzata tramite Three.js [3].

[1] `https://playground.babylonjs.com/#F41V6N`
[2] `https://threejs.org/examples/`
[3] `https://threejs.org/examples/?q=xr#webxr_xr_dragging`

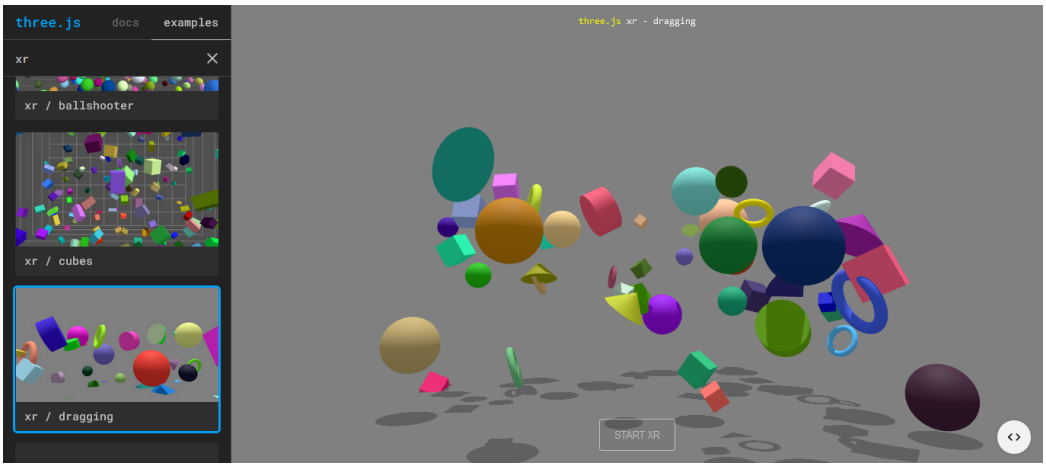


Figura 3.3: Esempio di applicazione WebXR realizzata tramite Three.js

Concludendo, per i requisiti visti prima, questi motori di rendering, non solo ci consentono di supportare differenti dispositivi grazie a WebXR, ma anche di **poter mostrare lo stesso contenuto aumentato ai diversi partecipanti** e risulta utile adottarli per la realizzazione dei nostri sistemi.

3.3 Croquet

Croquet è un sistema di sincronizzazione per esperienze digitali multiutente. Consente a più utenti di poter lavorare assieme all'interno di un unico ambiente distribuito, garantendo che questo ambiente sarà in grado di rimanere identico per ognuno di loro [4].

Per quanto abbiamo detto nel capitolo 2, quindi, Croquet rappresenta uno strumento molto utile per poter realizzare esperienze condivise, in cui riuscire a sincronizzare la visione del mondo dei diversi partecipanti, risulta essere un elemento cruciale, consentendoci di andare a soddisfare il requisito visto prima di: **consentire agli utenti di poter vedere le modifiche degli altri effettuati ad oggetti condivisi**.

La sincronizzazione effettuata da Croquet è in gran parte invisibile allo sviluppatore. La creazione di un'applicazione Croquet non richiede al programmatore di **scrivere alcun codice server-side o di rete**. Le applicazioni vengono strutturate in una parte **condivisa** ed una **locale**, ma entrambe le parti vengono eseguite localmente e sviluppate utilizzando solo strumenti *client-side*. La libreria di Croquet si occupa del resto [4].

3.3.1 Come funziona Croquet

Andiamo ad analizzare in dettaglio come funziona Croquet e quali sono i meccanismi che tale libreria mette in atto per poter ottenere la sincronizzazione che stavamo cercando.

Iniziando con una visione di alto livello, possiamo osservare, grazie alla figura 3.4 , come ai differenti utenti connessi alla mia applicazione sia associato uno stesso **computer virtuale** e l'insieme di computer virtuali, associati ai differenti partecipanti, formino un unico **computer logico** da cui ogni utente possiede una linea di input e una linea di output. Tutto quello che lo sviluppatore deve fare è programmare questo singolo computer virtuale [10].

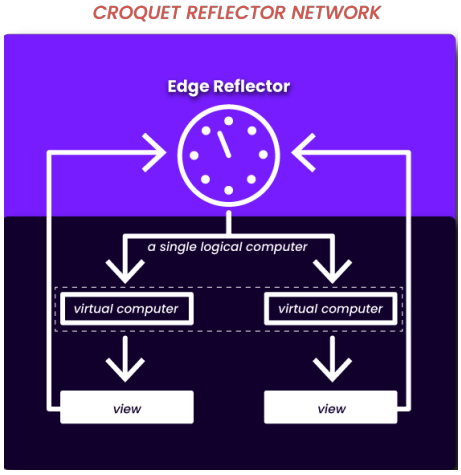


Figura 3.4: Diagramma di un'applicazione Croquet

Il meccanismo principale su cui si fonda il funzionamento di Croquet è la **sincronizzazione degli eventi**. Per poter comprendere meglio questo concetto, sempre considerando il diagramma in figura 3.4, consideriamo un esempio in cui la nostra applicazione coinvolga due utenti: l'utente di sinistra e l'utente di destra; supponendo ora che l'utente di sinistra effettui un'operazione di selezione, tale operazione genererà un evento, il quale verrà inviato a un componente di Croquet chiamato *Reflector*, che si occuperà di distribuire l'evento generato al Client dell'utente di destra, il quale ricevuto l'evento lo elaborerà eseguendo il relativo *handler*, riuscendo così ad andarsi a sincronizzare con l'operazione di selezione effettuata precedentemente.

I *refelctors*, messi a disposizione da Croquet, sono dei Server dispiegati nel Cloud, i quali si occupano semplicemente di "riflettere", come uno specchio , gli eventi che li arrivano proiettandoli sugli altri utenti collegati [4]. Tramite di essi riusciamo ad eseguire lo stesso aggiornamento in tutti i clients, senza che nessuna elaborazione venga effettuata lato Server o a livello di Reflector.

Inoltre, il Reflector di Croquet, detiene il **tempo virtuale** della nostra sessione e si occupa di attribuire un *timestamp* all'evento che è stato pubblicato per far sì che i clients, incaricati di riceverlo, lo elaborino tutti esattamente nello stesso istante. Il tempo che viene attribuito ai messaggi dal Reflector è **relativo al futuro**, infatti, i clients quando ricevono un evento, lo programmano per il futuro ed avendo condizione del tempo che passa, in quanto notificati del ticchettio dell'orologio detenuto dal Reflector, quando giunge il momento di elaborare l'evento precedentemente ricevuto, si occupano di eseguire l'*handler* ad esso associato.

3.3.2 Architettura di un'applicazione Croquet

Entrando in dettaglio del funzionamento di un'applicazione Croquet, notiamo che per poter mantenere i *virtual computers* dei singoli utenti sincronizzati, generando un unico computer logico, la strategia che viene adottata prevede di **dividere il nostro programma in due componenti** principali:

- **Model**, il quale si occupa di gestire tutti i calcoli e le simulazioni, ed è all'interno di esso dove avviene il lavoro effettivo dell'applicazione;
- **View**, incaricato di gestire gli input e gli output degli utenti elaborando tutti gli eventi di tastiera, mouse, *touch* che vengono generati e di determinare ciò che viene visualizzato sullo schermo.

Lo stato del Model è sempre garantito essere **identico** per tutti gli utenti, tuttavia lo stato della View non lo è, in quanto, utenti diversi potrebbero utilizzare hardware differenti per poter accedere al contenuto offerto, ed è comunque possibile mostrare agli utenti differenti rappresentazioni della simulazione [5].

Le **comunicazioni interne** che avvengono fra il Model e la View vengono **gestite per mezzo degli eventi**, pertanto, ogni qual volta un oggetto pubblica un evento, tutti i sottoscrittori ad esso riceveranno la notifica e potranno procedere all'esecuzione del relativo handler associato [5].

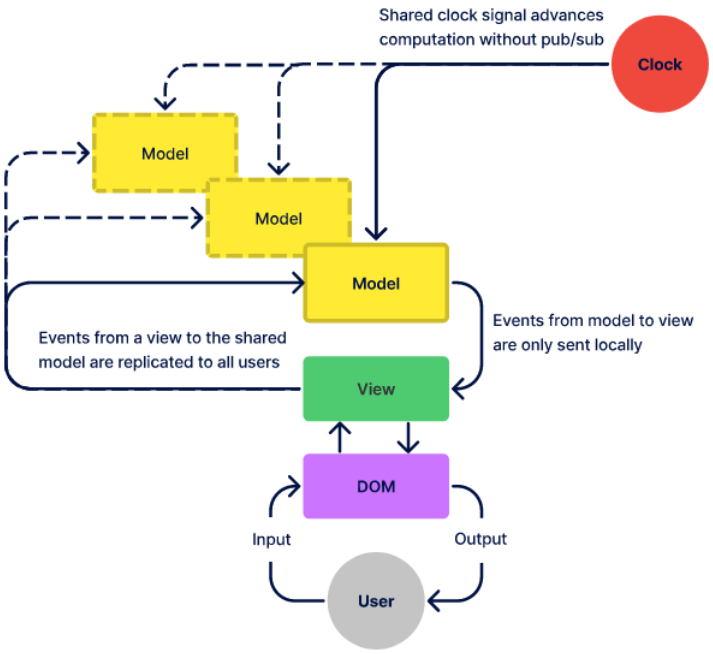


Figura 3.5: Diagramma di un'applicazione Croquet

Come si può notare dalla figura 3.5, gli eventi generati **non sempre** necessitano di essere inoltrati tramite il Reflector. Infatti, come sappiamo, ad ogni utente viene associato un Model e una View e gli eventi generati dal Model vengono **inviati solo localmente** alla View, senza passare per il Reflector, perché se è vero che lo stato del Model è garantito essere sincronizzato fra tutti gli utenti, allora tutti i Model invieranno il medesimo evento (nello stesso momento) alla View a loro collegata. Mentre gli eventi generati dalla View verranno inviati a tutti i Model associati ai diversi utenti, per mezzo dei reflectors, in modo da mantenerli sempre sincronizzati sulle modifiche che sono avvenute.

Fra questi due elementi principali che abbiamo descritto, è presente un terzo componente chiamato **Controller**, il quale si occupa di indirizzare gli eventi tra il Model e la View, tramite il Reflector se necessario.

Sebbene l'architettura realizzata da Croquet, coinvolga questi tre elementi: Model, View e Controller (figura 3.6), il pattern seguito non è MVC [10], in quanto le caratteristiche che questi elementi presentano, rispetto a tale pattern, sono differenti e per ora, gli autori di Croquet, non hanno ancora saputo attribuire un nome appropriato a questo nuovo approccio utilizzato.

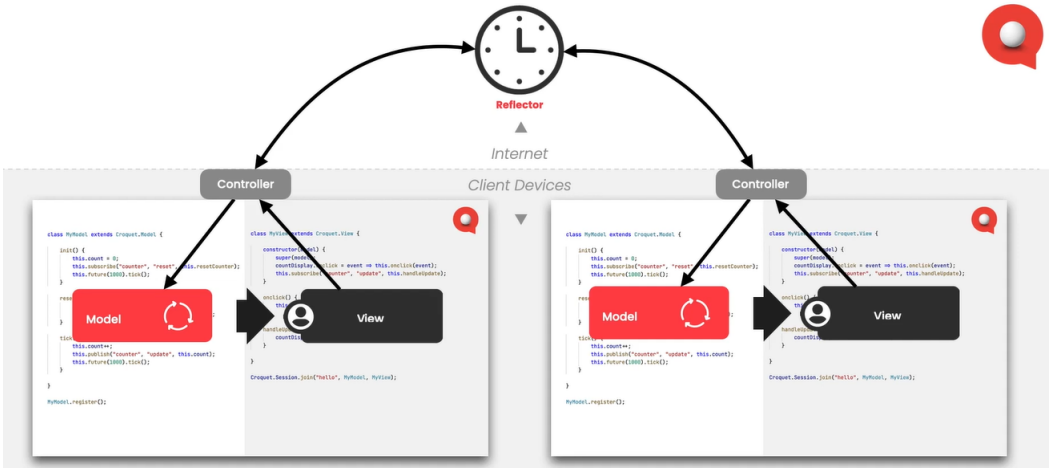


Figura 3.6: Diagramma architettura Model-View-Reflector di Croquet

Alla luce di quanto detto fino ad adesso, facendo riferimento sempre allo schema di figura 3.6, quando un'utente esegue un'iterazione sulla View, possiamo riassumere con il seguente elenco i passi seguiti dalla nostra applicazione:

1. La View su cui si è verificata l'iterazione pubblica un evento, con un timestamp vuoto;
2. Tale evento viene gestito dal Controller, il quale è a conoscenza del fatto che registrati a quel determinato evento vi siano dei modelli e di conseguenza, decide di inoltrare l'evento ricevuto per mezzo di un *Reflector*;
3. Quando l'evento raggiunge il *Reflector* esso gli assegna un timestamp e si occupa del suo inoltro ai diversi clients connessi;
4. I modelli dei clients connessi ricevuto l'evento eseguono il relativo handler, sincronizzandosi con la modifica apportata.

Dalla analisi dei precedenti passi, possiamo far emerge un'osservazione importante. Il Reflector, attribuendo egli stesso il timestamp agli eventi che sono stati pubblicati, fa sì che tutti **i modelli elaborino l'evento nello stesso momento**, di conseguenza, gli aggiornamenti che vengono effettuati dal Model, vengono scanditi da un **orologio**, il quale consente che due eventi che arrivano al Reflector, uno di seguito all'altro, vengano etichettati con differenti timestamp considerando il tempo trascorso.

Il ticchettio dell'orologio adottato dai Reflector, è **deterministico** [10] il che consente di far avanzare lo stato del Model in modo corretto, garantendo la sincronizzazione fra i diversi utenti, portandoci ad avere l'effetto che volevamo.

Capitolo 4

Un framework per la realizzazione di applicazioni di eXtended Reality collaborative

Dopo aver illustrato i diversi strumenti che possono essere adottati per costruire le nostre applicazioni, possiamo passare all'analisi del framework realizzato, il quale ci consentirà di raggiungere il nostro obiettivo: realizzare applicazioni di eXtended Reality collaborative.

Innanzitutto, chiariamo che cosa si intende nel nostro caso per framework. Per framework intendiamo uno strumento di sviluppo in grado di predisporre un ambiente con alcune caratteristiche, in questo caso si prevede di fornire un ambiente che supporti l'eXtended Reality collaborativa, che possa essere utilizzato dal programmatore per poter costruire determinate applicazioni, dandoli anche modo di estendere le sue funzionalità a partire dai componenti offerti, creandone dei nuovi.

Nel seguente capitolo, pertanto, verrà descritto il framework creato partendo da un'analisi dei requisiti: funzionali, non funzionali, utente e di implementazione individuati per esso, per poi passare a un'analisi della sua architettura prima con un ottica di alto livello, descrivendone i componenti principali, poi entrando nel dettaglio dei diversi elementi realizzati, delle funzionalità messe a disposizione e di come queste siano state ottenute.

4.1 Requisiti

Per il framework realizzato sono stati valutati i seguenti tipi di requisiti:

- **funzionali**, riguardano le funzionalità che il sistema deve mettere a disposizione dell'utente;

- **non funzionali**, riguardano le funzionalità che il sistema non deve necessariamente possedere per fare in modo che sia funzionante e corretto;
- **utente**, esprimono i bisogni degli utenti e descrivono quali sono le azioni che l'utente deve poter effettuare interagendo con il sistema;
- **di implementazione**, si tratta di requisiti che vincolano l'intera fase di realizzazione del sistema, ad esempio, richiedono l'uso di uno specifico linguaggio di programmazione e/o di uno specifico strumento software.

Requisiti funzionali

I requisiti funzionali individuati sono i seguenti:

- creare un mondo aumentato in cui poter inserire degli elementi olografici;
- gestire l'interazione con l'utente tramite l'adozione di opportune interfacce grafiche;
- consentire la manipolazione degli elementi olografici presenti nel mondo, potendone modificare le loro proprietà in modo dinamico o statico;
- dare la possibilità agli utenti di poter vedere le manipolazioni dello stato degli elementi, effettuate da lui o da altri.

Requisiti non funzionali

I requisiti non funzionali analizzati per il framework, sono i seguenti:

- deve essere sufficientemente modulare, in modo tale che una modifica a un componente non richiedi di aggiornare un numero elevato di elementi;
- deve essere in grado di scalare a seguito dell'aggiunta o rimozione di nuove funzionalità;
- deve essere in grado di gestire le problematiche che possono emergere avendo a che fare con un sistema distribuito quali: corse critiche, consistenza degli aggiornamenti, resistenza ai guasti etc.

Requisiti utente

Per quanto riguarda i requisiti utente, per noi l'utente di riferimento è il programmatore, pertanto, sono stati individuati i seguenti requisiti:

- il framework deve essere semplice da utilizzare, non deve rendere più complicata di quanto non lo sia già ora, la realizzazione di questo tipo di sistemi;
- l'utente deve avere la possibilità di modificare gli elementi offerti o aggiungerne di nuovi in modo agevole, se ad esempio, desidera cambiare una specifica tecnologia di riferimento, deve poterlo fare intaccando il minor numero di elementi possibile.

Requisiti di implementazione

I requisiti implementativi individuati per il framework, sono di seguito riportati:

- deve fornire un supporto ai dispositivi per tutti i differenti tipi di realtà, in modo da consentire la realizzazione di esperienze di eXtended Reality;
- deve essere programmato attraverso il linguaggio *Javascript* e utilizzare moduli *ES6* per la definizione dei diversi componenti;
- deve poter essere eseguibile su tutti i differenti tipi di browser che forniscono un supporto a WebXR, illustrati in figura 4.1.

	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android
xr	✓ 79	✓ 79	✗ No	✓ 66	✗ No	✓ 79	✗ No	✓ 57	✗ No	✓ 11.2	✗ No

Figura 4.1: Browser che supportano WebXR

4.2 Design ad alto livello

L'architettura del framework può essere scomposta in due parti principali: una componente **Core** e una componente **Infrastructure** (figura 4.2).

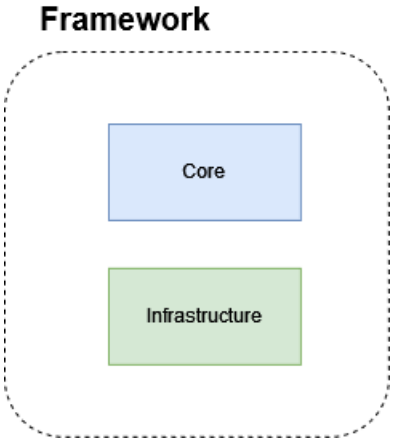


Figura 4.2: Architettura ad alto livello

La componente Core racchiude tutti gli elementi principali del framework che si occupano di descrivere qual'è l'effetto che lo sviluppatore vuole ottenere e sono quelli con cui il programmatore interagirà direttamente; essi però **non racchiudono** all'interno alcun elemento infrastrutturale, vale a dire alcuna tecnologia o strumento esterno, che specifichi **come** l'effetto desiderato verrà ottenuto, ma definiscono solamente che **cosa** si vuole andare ad ottenere.

La componente Infrastructure, invece, racchiude tutti quegli aspetti implementativi e strutturali che vengono messi in atto, attraverso l'utilizzo di apposite tecnologie e strumenti, per realizzare quanto l'utente ha richiesto.

Gli elementi che si trovano nella parte Core del framework hanno la possibilità di interagire con gli elementi della parte Infrastructure, per mezzo degli **eventi**. Infatti, all'interno del framework, è presente un `EventBus`, il quale dà la possibilità ai componenti delle due parti principali di poter comunicare fra loro, come si può vedere da figura 4.3.

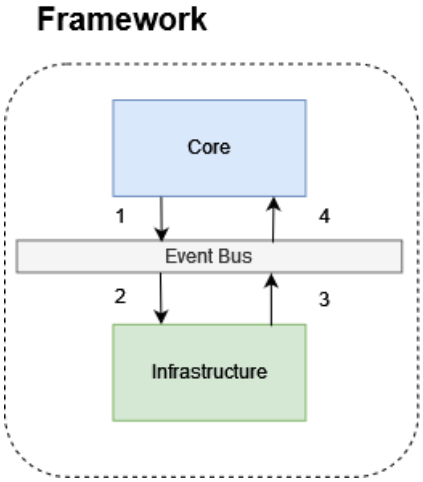


Figura 4.3: Ordine generale degli eventi inviati e ricevuti

Relativamente a quest'aspetto, un'osservazione che è opportuno fare è che l'interazione che si viene a creare viene sempre fatta partire da un componente della parte Core, il quale poi potrà attendere la risposta dalla parte Infrastracture, che avverrà mediante la pubblicazione di un apposito evento. Ad ogni modo, se la parte Core **non manda** alcun tipo di evento, all'interno dell'applicazione **non accadrà nulla**; è questa componente, infatti, che **impartisce ordini** all'altra e si aspetta di ottenere determinati risultati.

Per capire meglio questo aspetto, analizzando la figura 2.11 possiamo osservare quali siano i passaggi che avvengono durante l'interazione: si parte sempre da un elemento nella parte Core, il quale pubblicherà un evento (1), che verrà recepito da uno racchiuso all'interno della parte Infrastracture (2) che può richiedere di restituire una risposta, la quale verrà inviata sempre da un elemento appartenente a Infrastructure (3) e verrà ricevuto da un entità di Core (4) predisposta alla sua ricezione e gestione.

4.2.1 Componenti principali

Gli elementi principali che il programmatore si troverà ad utilizzare si trovano all'interno della parte Core e sono riportati dalla seguente figura 4.4.

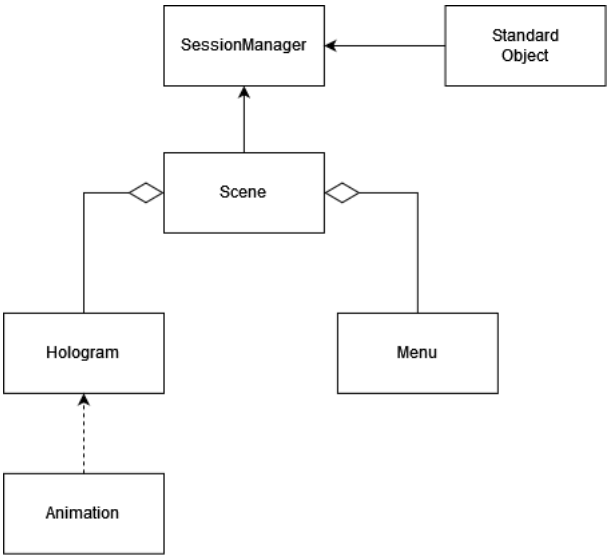


Figura 4.4: Componenti principali dell'architettura della parte Core

La **scena** rappresenta il mondo visto dal partecipante all'esperienza. Ad ogni scena è associata una **sessione**, che consente a più utenti di potersi connettere e vedere lo stesso contenuto. Quello che il programmatore si occuperà di fare, sarà andare a costruire la scena che si vuole mostrare all'utente, per poter fare questo, i due elementi principali del framework che verranno utilizzati saranno: gli ologrammi e i menu.

Le proprietà degli **ologrammi** possono essere **sincronizzate** e pertanto, ogni modifica che verrà effettuata su di loro, verrà percepita da tutti i partecipanti coinvolti all'esperienza, volendo però queste proprietà possono anche essere modificate senza sincronizzazione, in modo da produrre un effetto diverso per i differenti partecipanti. Essi ci rappresentano elementi del mondo che vogliamo mostrare all'utente, che hanno una propria locazione nello spazio e con cui l'utente può avere la possibilità di interagire.

I **menu**, invece, rappresentano gli elementi dell'interfaccia grafica con cui il partecipante all'esperienza può relazionarsi, essi sono gestiti in modo indipendente per ogni utente, il che significa che ogni utente avrà una **propria visione** dei menu e potrà interagire con questi indipendentemente dagli altri, compiendo su di questi azioni che non andranno ad intaccare quelli degli altri partecipanti.

Avendo fatto questa distinzione, fra ologrammi e menu, occorre puntualizzare che, essendo un ologramma un oggetto fatto di luce, il quale può essere fatto apparire nello spazio attorno a noi [27], risulta che anche i **menu sono di fatto ologrammi**. Infatti, essi hanno una propria rappresentazione olografica, che li presenta all'utente come oggetti fatti di luce. Tuttavia, te-

niamo questa distinzione: ologrammi e menu, per poter meglio **discernere** gli **elementi** che fanno parte dell'**interfaccia grafica** da quelli che non lo sono, prendendo a riferimento quello che anche Microsoft segue nei diversi esempi e tutorial, utilizzati per insegnare agli sviluppatori come realizzare applicazioni di Mixed Reality, adottando per riferirsi agli elementi dell'interfaccia grafica il loro nome specifico (es. `NearMenu`) e agli ologrammi il loro nome (es. `Cube`) o il termine generico ologramma (e.g [23][26]).

Per poter attribuire un **comportamento dinamico** agli ologrammi, come ad esempio un movimento o un cambio di colore continuo, possono essere adottate le **animazioni**, le quali ci consentono di poter andare a modificare, con una certa frequenza, alcune proprietà degli elementi della scena, producendo un determinato effetto visivo.

Infine, gli **oggetti standard**, indicati nella figura 4.4 con il termine di `StandardObject`, ci rappresentano degli oggetti semplici caratterizzati da un unica proprietà che può essere aggiornata in modo sincrono oppure no e che possono essere utilizzati per poter gestire particolari aspetti della nostra applicazione, ad esempio un contatore condiviso; ci rappresentano strumenti che possono essere **utilizzati direttamente dal programmatore**, per poter avere un oggetto caratterizzato da un valore/proprietà, che può essere uguale per tutti i clients connessi, anche a seguito di modifiche che possono essere effettuate su di esso. Agli oggetti standard **non** è associato alcun elemento grafico o effetto visivo essi, infatti, non si riferiscono alla scena, ma alla sessione creata e sarà questa che si occuperà di gestire la loro creazione.

Questi, quindi, sono i componenti che lo sviluppatore ha a disposizione per poter realizzare la sua applicazione, nelle successive sezioni andremo ad analizzare con maggiore dettaglio l'architettura del framework, andando a mettere in luce anche gli elementi con cui il programmatore non si relaziona direttamente, ma che sono necessari per poter far funzionare correttamente il nostro sistema. Nel design di dettaglio verranno comunque ripresi anche i componenti qui descritti, indicandone le loro caratteristiche e effettuandone una descrizione maggiormente dettagliata (si veda la sezione 4.3).

4.3 Design di dettaglio parte Core

Passiamo a descrivere in dettaglio l'architettura della parte Core, partendo dall'analisi dell'elemento `Scene` e di altri componenti con con cui esso interagisce per poi passare agli elementi: `Hologram`, `Animation` e `SynchronizedObject`.

4.3.1 Scene

La componente più importante della parte Core è rappresentata dalla classe `Scene`, le cui caratteristiche sono riportate in figura 4.5.

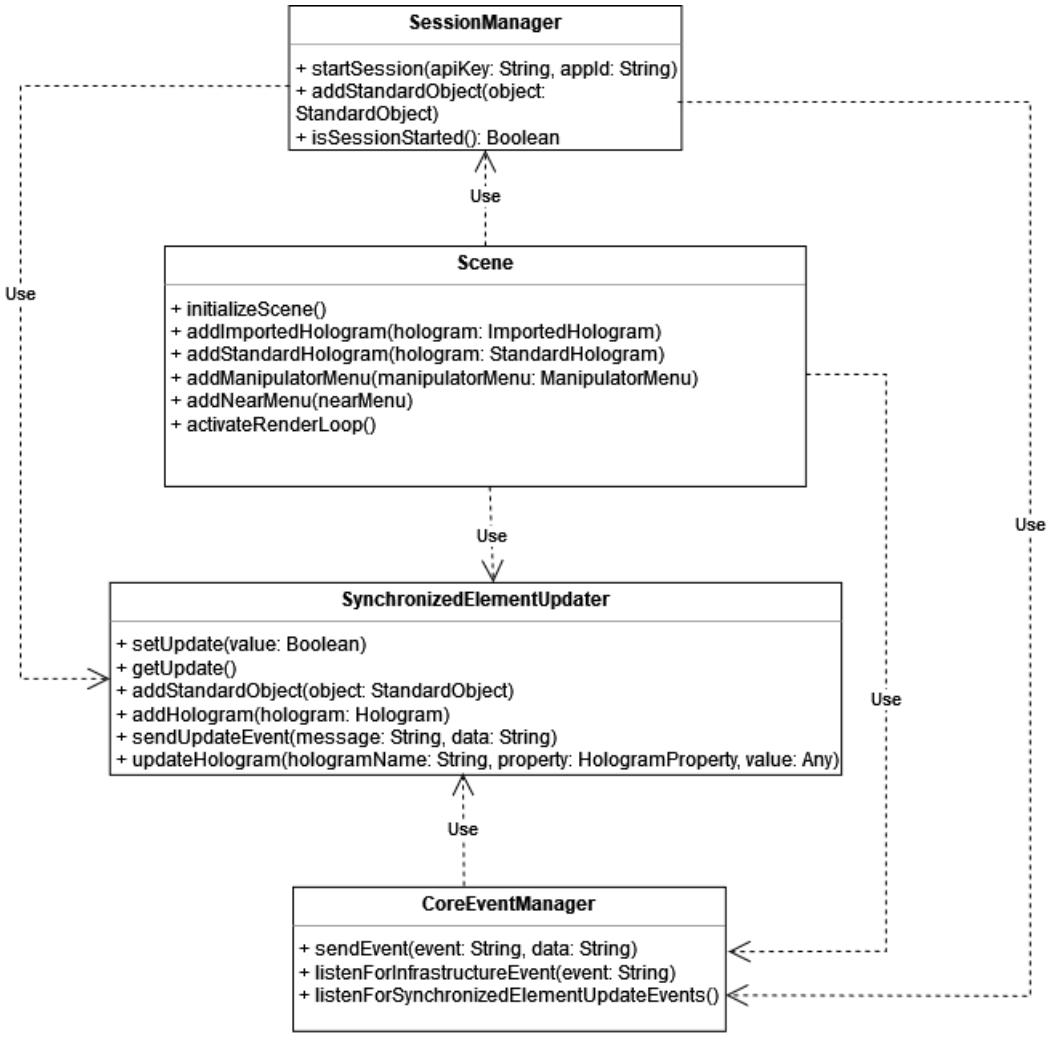


Figura 4.5: Diagramma delle classi: Scene, SessionManager, SynchronizedElementUpdater e CoreEventManager

Come precedentemente annunciato, l'elemento `Scene` è il principale componente con cui l'utilizzatore del framework interagirà, consentendoli di aggiungere gli elementi desiderati alla scena in modo da poter ottenere l'effetto voluto.

Tuttavia, per poter inserire gli elementi nella scena correttamente, lo sviluppatore deve rispettare **due vincoli** principali:

1. Il primo, richiede l'avvio di una sessione Croquet per mezzo dell'elemento `SessionManager`, specificando un'apposita `apiKey` e un `AppId`, i quali possono essere ottenuti a seguito dell'iscrizione al portale ufficiale di Croquet[1].

2. Il secondo, richiede che prima di inserire elementi all'interno della scena, questa venga inizializzata, attraverso l'esecuzione del metodo `initializeScene`.

Una volta rispettati questi due passaggi, l'utente avrà la possibilità di aggiungere alla scena, tutti gli elementi che li serviranno, specificando le loro proprietà attraverso apposite classi (descritte in seguito) e procedere all'avvio del *render-loop*.

Il seguente listato 4.1 riassume i passaggi che il programmatore deve seguire per renderizzare correttamente una scena.

```
//imports
const apiKey = "myApiKey";
const appId = "myAppId";

const sessionManager = new SessionManager();
await sessionManager.startSession(apiKey, appId);
const scene = new Scene(sessionManager);
scene.initializeScene();

/**
* Add elements to scene...
*/

scene.activateRenderLoop();
```

Listato 4.1: Inizializzazione di una scena

Sempre osservando la figura 4.5, possiamo vedere come l'elemento `Scene`, sfrutti due classi di utilità chiamate: `CoreEventManager` e `SynchronizedElementUpdater`.

`CoreEventManager` racchiude l'`EventBus` con cui vengono gestiti gli eventi. Difatti, come abbiamo spiegato prima, i componenti della parte Core mostrano all'utente che cosa si può fare e non contengono alcuna informazione su come le cose verranno fatte, questo spetta alla parte Infrastructure stabilirlo; pertanto, la componente `Scene` attraverso `CoreEventManager`, il quale si occuperà di gestire gli eventi da inviare e da ricevere alla parte Infrastructure, sarà in grado

[1] `https://croquet.io/account/`

di impartire degli ordini a tale parte, inviando un apposito evento tramite questa classe.

`SynchronizedElementUpdater`, invece, detiene al suo interno tutti gli elementi che risultano essere sincronizzati fra i diversi utenti (come gli ologrammi) e si occupa di **mantenere consistente il loro stato** a seguito delle modifiche che possono avvenire per mezzo dell'interazione con l'utente o previste dal comportamento che ci si aspetta dall'elemento utilizzato, le quali verranno notificate dalla parte Infrastructure.

4.3.2 Hologram

Il framework realizzato da la possibilità all'utente di poter creare e aggiungere all'applicazione degli elementi, le cui proprietà possono essere sincronizzate per i diversi partecipanti all'esperienza, consentendo a tutti i clients collegati di poter osservarne lo stesso stato. Tali elementi includono gli ologrammi rappresentati mediante la classe `Hologram`.

La classe `Hologram` ci rappresenta un'entità presente nel mondo, avente una propria **posizione** all'interno della rappresentazione dello spazio mantenuta dal sistema e una propria resa grafica.

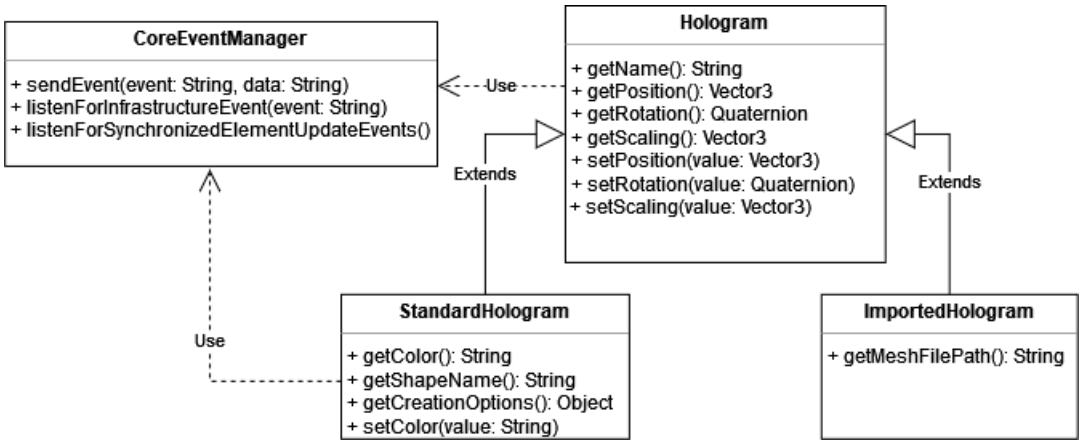


Figura 4.6: Diagramma delle classi: Hologram, StandardHologram e ImportedHologram

Come si può vedere dalla figura 4.6 vi sono due diverse tipologie di ologrammi espressi attraverso la definizione di due diverse classi: `StandardHologram` e `ImportedHologram`.

Entrambi, `StandardHologram` e `ImportedHologram`, ereditano le proprietà di `Hologram` e la **differenza** sostanziale che lega queste due classi è relativa alla **resa grafica degli elementi**. In un caso, questa è definita tramite un

apposito file ad esempio in formato `.gltf` o `.glb`, nell'altro, invece, la forma dell'ologramma è definita a partire da una figura specifica come una sfera, un cubo, un piano etc. che per poter essere realizzata non richiede l'importazione di alcun tipo di file, in quanto, le forme standard che gli ologrammi possono assumere sono già state definite all'interno del framework.

Altre osservazioni che possiamo fare guardando il diagramma delle classi (figura 4.6) sono le seguenti:

- oltre alle informazioni per la loro rappresentazione gli ologrammi presentano anche un'altra proprietà che è il **nome**, questo in quanto all'interno del framework gli elementi vengono riconosciuti attraverso questo attributo, di conseguenza, esso ci rappresenta un **identificativo univoco** che può essere utilizzato per poter ottenere un loro riferimento;

- gli ologrammi utilizzano l'elemento `CoreEventManager` per poter informare la parte infrastrutturale degli aggiornamenti avvenuti, attraverso l'invio di appositi eventi.

Una volta creato l'ologramma, specificando le sue proprietà, questo può essere aggiunto alla scena attraverso la componente `Scene`, come mostrato dal listato 4.2.

```
//...
const name = "Sphere";
const position = new Vector3(0, 1, 1.2);
const rotation = new Quaternion(0,0,0,0);
const color = "#fbff2b";
const creationOptions = {diameter: 0.5}
const sphere = new StandardHologram(hologramName,
    StandardShape.Sphere, creationOptions, hologramPosition,
    hologramRotation, hologramColor);

await scene.addStandardHologram(sphere);
scene.activateRenderLoop();
```

Listato 4.2: Aggiunta di un ologramma alla scena

Attraverso questo esempio, possiamo notare che fin quando l'elemento creato non viene aggiunto alla scena, egli **non sarà visibile all'utente**, esso esiste in quanto è stato creato, ma la sua rappresentazione grafica non apparirà al partecipante, fin tanto che questi non verrà aggiunto alla scena.

4.3.3 StandardObject

Un ulteriore elemento sincronizzato che il programmatore può richiedere di utilizzare, sono gli "oggetti standard" rappresentati dalla classe `StandardObject` (figura 4.7).

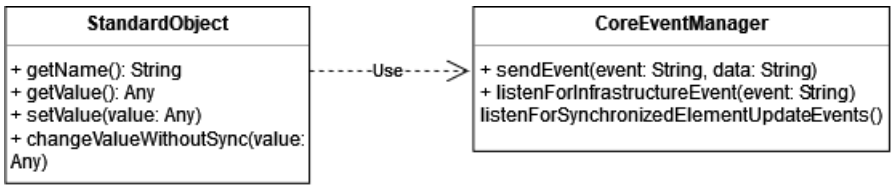


Figura 4.7: Diagramma delle classi: SynchronizedVariable

Anche gli oggetti standard, come gli ologrammi, presentano un nome che ci consente di riferirci ad essi in modo univoco e utilizzano l'elemento `CoreEventManager` per poter informare la parte infrastrutturale del loro cambio di valore, se si vuole che questa modifica venga percepita anche dagli altri utenti.

Tuttavia, la differenza sostanziale che vi è fra questi elementi e gli ologrammi, è che per gli oggetti standard non è prevista alcuna rappresentazione grafica. Pertanto, questi oggetti, vengono **utilizzati direttamente nel codice di programmazione**, per poter gestire determinate funzionalità come, ad esempio, le animazioni (vedi listato 4.4).

Ad ogni modo, lo stato dell'oggetto nel caso di modifiche che si vuole siano sincronizzate, viene mantenuto consistete tramite il componente `SynchronizedElementUpdater`, come visto per gli ologrammi.

Per poter aggiungere una oggetto standard alla sessione, occorre richiamare il metodo `addStandardObject` di `SessionManager` (figura 4.5), questa volta, quindi, come accennato precedentemente, non aggiungeremo un oggetto passando per il componente `Scene`, in quanto questo nulla ha a che fare con la scena, non è un componente che possiede alcun elemento grafico che debba essere mostrato all'utente, ma esso riguarda la sessione che abbiamo creato e pertanto, per poterla aggiungere e utilizzare correttamente garantendo la sua effettiva sincronizzazione, egli deve essere aggiunto alla sessione passando per `SessionManager`, altrimenti non lo si potrà aggiornare in modo opportuno.

Nel seguente listato 4.3 è possibile vedere i passaggi descritti prima e come sia possibile modificare, in modo sincronizzato, il valore detenuto da uno `StandardObject`.

```
//...
const name = "counter";
const initialValue = 0;
```

```
const counter = new StandardObject(name, initialValue);
sessionManager.addStandardObject(counter);

counter.value = counter.value + 1;

scene.activateRenderLoop();
```

Listato 4.3: Aggiunta e modifica del valore di una variabile sincronizzata

4.3.4 Animation

Un altro elemento che è possibile realizzare tramite il framework sono le animazioni, rappresentate mediante la classe `Animation`.

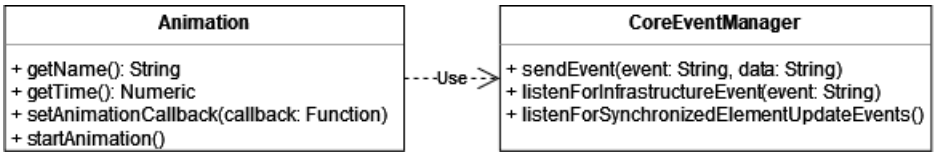


Figura 4.8: Diagramma delle classi: Animation

Come possiamo vedere dalla figura 4.8, un'animazione è caratterizzata da: un nome, un tempo di *scheduling* e una callback da richiamare quando il tempo è scaduto.

La componente `Animation`, rientra fra gli elementi sincronizzati della parte Core, dato che il tempo specificato per essa è **uguale per tutti** i partecipanti. Esso, infatti, fa riferimento alla sessione creata e la callback specificata verrà richiamata per tutti, anche per i partecipanti che si aggiungeranno in seguito all'esperienza, allo stesso momento, in modo che tutti gli utenti siano in grado di vedere lo stesso stato dell'entità a cui l'animazione è associata. Per poter consentire questo, un oggetto della classe `Animation`, richiede a `CoreEventManager` di mettersi **in ascolto di un determinato evento** emesso dalla parte infrastrutturale, il quale informa che il tempo specificato per l'animazione è trascorso.

Il seguente listato 4.4 mostra come può essere creata ed impostata un animazione. Per prima cosa occorre specificare il nome dell'animazione e impostare il tempo di quest'ultima in millisecondi, fatto questo, è possibile creare l'animazione attraverso la classe `Animation`, a questo punto si può impostare la callback che vogliamo venga richiamata ogni secondo, infine, per poter vedere l'effetto prodotto dall'animazione, occorre farla partire tramite la chiamata al metodo `startAnimation`.

```
//...
const animationTime = 1000;
const animationName = "changeColor";
const changeColor = new Animation(animationName, animationTime);

changeColor.setAnimationCallback(() => {
    counter.value = counter.value + 1;
    if((i.value % 2) === 0) {
        sphere.color = "#ff5e00"
    }else{
        sphere.color = "#00fbff"
    }
});

changeColor.startAnimation();

scene.runRenderLoop();
```

Listato 4.4: Aggiunta di un'animazione

Anche in questo caso, possiamo notare, come l'attributo nome venga utilizzato per poter riferirci all'animazione in modo univoco e anche che l'animazione creata verrà eseguita continuamente. Ogni secondo la callback viene richiamata, fino a quando non si decide di fermare l'animazione attraverso la chiamata al metodo `stopAnimation`.

4.3.5 Menu

Il framework realizzato da anche la possibilità all'utente di poter inserire nella scena delle interfacce grafiche, con cui il partecipante all'esperienza potrà interagire.

In particolare, gli ologrammi inseriti nella scena, possono essere manipolati dagli utenti, questo significa che il partecipante potrà interagire con loro come farebbe con un oggetto reale:"toccandolo", attraverso lo specifico strumento di interazione previsto per il dispositivo adottato e il tipo di realtà di riferimento.

Per consentire a un partecipante la manipolazione di un ologramma, il progettista dell'applicazione deve richiedere l'aggiunta di un `ManipulatorMenu` (figura 4.9) da associargli. Le modifiche effettuate alle proprietà dell'ologramma attraverso il `ManipulatorMenu` sono **sincronizzate fra tutti i partecipanti all'esperienza per default**, se un'utente sposta l'ologramma a sinistra, esso si muoverà anche per tutti gli altri partecipanti e le sue coordinate saranno uguali per tutti, pertanto, i diversi utenti vedranno l'ologramma nello stesso

punto della rappresentazione dello spazio mantenuta dal sistema, tramite il dispositivo adottato.

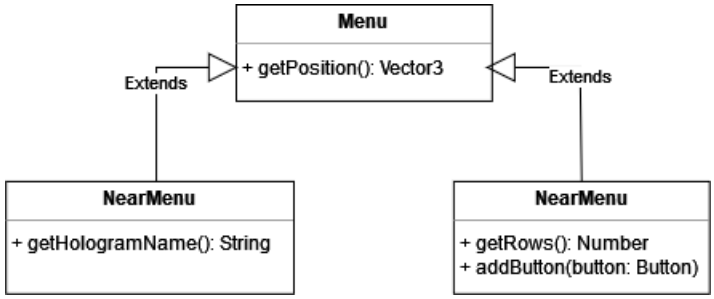


Figura 4.9: Diagramma delle classi: Menu, NearMenu e ManipulatorMenu

Quando si richiede di aggiungere un `ManipulatorMenu`, occorre specificare la posizione nello spazio in cui vogliamo che questo appaia e il nome dell'ologramma a cui fa riferimento. Infine, per poter aggiungere effettivamente il menu e renderlo visibile all'utente, come per gli ologrammi, occorre passare per l'elemento `Scene`.

Il seguente listato 4.5 mostra come sia possibile associare un menu di manipolazione, a un dato ologramma.

```
//...
const manipulatorMenuPosition = new Vector3(-0.3, 1.2, 0.5)
const manipulatorMenu = new ManipulatorMenu(manipulatorMenuPosition,
    "sphere");
scene.addManipulatorMenu(manipulatorMenuHologram3);

scene.activateRenderLoop();
```

Listato 4.5: Aggiunta di un ManipulatorMenu

La figura 4.9 ci mostra, inoltre, che esiste un ulteriore tipologia di menu che può essere aggiunta alla scena, identificata dalla classe `NearMenu`.

La differenza sostanziale che lega `ManipulatorMenu` e `NearMenu` riguarda il fatto che, il primo rappresenta un **menu di default** stabilito dal framework, il quale svolge quanto promesso, cioè consentire la manipolazione dell'ologramma a lui associato. Il secondo, invece, ci rappresenta un menu che può essere maggiormente **impostato dal progettista**, egli infatti potrà decidere: in quante righe è organizzato il menu, di quanti bottoni si compone e qual'è la callback da richiamare se uno di questi viene premuto.

Button

Per poter configurare correttamente un `NearMenu` , occorre creare e definire il comportamento dei bottoni che ne andranno a fare parte, attraverso l'utilizzo della classe `Button` (figura 4.10).

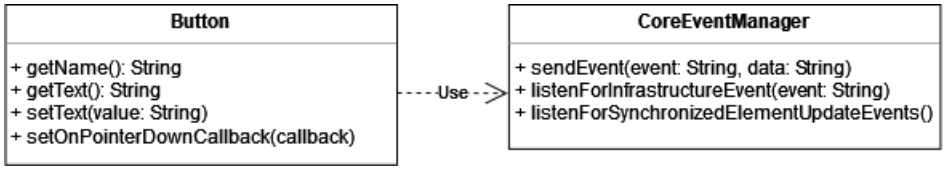


Figura 4.10: Diagramma delle classi: Button

Per ogni bottone creato viene specificata una callback da richiamare quando il pulsante viene premuto. Il bottone creato per sapere se è stato effettivamente premuto dall'utente, tramite `CoreEventManager`, si metterà in ascolto di uno specifico evento, proveniente dalla parte infrastrutturale, che lo informerà di questo avvenimento.

Per capire meglio come utilizzare un menu di questo tipo, osserviamo il seguente listato 4.6, che ci mostra come poterne creare uno costituito da due bottoni. Per prima cosa, occorre specificare il numero di righe su cui i bottoni saranno organizzati, nel nostro caso due righe, quindi, nel nostro menu i bottoni saranno disposti in verticale, se invece avessimo specificato un unica riga, i bottoni sarebbero stati organizzati in orizzontale; dopodiché bisogna specificare la posizione del menu, fatto questo, possiamo creare il `NearMenu` utilizzando la relativa classe. Infine, aggiungiamo al menu i bottoni da noi definiti, tramite cui il colore dell'ologramma `sphere` cambierà diventando blu, premendo il bottone uno e rosso, premendo il bottone due.

```
//...
const nRowsMenu = 2
const nearMenuPosition = new Vector3(0.3, 1.2, 0.5);
const nearMenu = new NearMenu(nearMenuPosition, nRowsMenu);

const button1 = new Button("button1", "Blue");
button1.setOnPointerDownCallback(() =>{
    sphere.color = "#0000ff"
});

const button2 = new Button("button2", "Red");
button2.setOnPointerDownCallback(() =>{
    sphere.color = "#ff0000"
});
```

```
nearMenu.addButton(button1);
nearMenu.addButton(button1);

scene.addNearMenu(nearMenu);

scene.runRenderLoop();
```

Listato 4.6: Aggiunta di un NearMenu

Possiamo notare anche qui, che per poter visualizzare correttamente il menu occorre passare per il componente `Scene`, altrimenti questo non potrà essere visto dai partecipanti. Inoltre, anche i bottoni realizzati, come già altri elementi che abbiamo visto, richiedono che venga specificato un nome per la loro creazione, in modo da riuscire a riferirci ad essi in maniera univoca.

4.4 Design di dettaglio della parte Infrastructure

Nella parte Infrastrucutre, come abbiamo detto, sono racchiusi tutti quegli aspetti tecnici e tecnologici che ci consentono effettivamente di produrre l'effetto desiderato dal progettista.

Come per la precedente parte, anche questa componente è costituita da un `InfrastructureEventManager`, il quale si occupa di gestire gli eventi inviati dalla parte Core e richiedere alla `RootView` l'esecuzione di specifiche operazioni.

In particolare, avendo scelto come tecnologia per gestire la sincronizzazione degli elementi: Croquet [4] (sezione 3.3), un'applicazione che sfrutti il sistema operativo, fornito da questo framework, presenta un'architettura ben definita costituita da un Model e almeno una View.

Di conseguenza, nella parte infrastrutturale sono presenti un `RootModel` e una `RootView`, i quali ereditano le proprietà definite per le classi di Croquet (`Croquet.Model` e `Croquet.View`), consentendoci di gestire gli elementi nella scena.

4.4.1 Elementi della View

La componente View del nostro sistema (figura 4.11) si occupa di gestire l'**interazione con l'utente** e le **richieste** del **programmatore**. Gli elementi di cui è costituita sono:

- una `RootView`, che ci rappresenta la View principale, la quale viene istanziata non appena un nuovo Client si connette e si occupa di gestire le

relazioni con il `RootModel` ed i menu per l'interazione con l'utente, che **non** hanno a che fare con la manipolazione degli ologrammi;

- una `HologramView`, la quale ci rappresenta una *sub-view*, istanziata da quella principale, incaricata di gestire tutti gli aspetti di visualizzazione degli ologrammi e la loro manipolazione, sulla base di quanto richiesto dal programmatore e quanto compiuto dal partecipante all'esperienza.

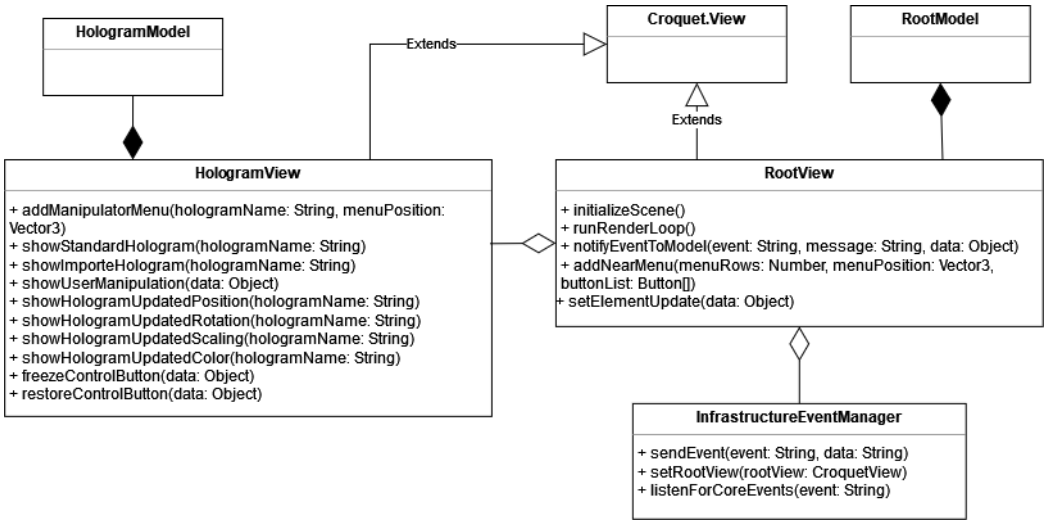


Figura 4.11: Diagramma delle classi: RootView e HologramView

Tutti gli eventi provenienti dalla parte Core, tramite `InfrastructureEventManager`, verranno gestiti dalla `RootView` la quale, se necessario, si occuperà di avvisare il `RootModel` dell'accadimento di un fenomeno attraverso la pubblicazione di un evento di Croquet.

Riguardo a questo, occorre precisare, che fra Model e View **non vi è alcuna comunicazione diretta**, essi comunicano fra loro scambiandosi degli eventi gestiti dal sistema operativo di Croquet, come spiegato nella sezione 3.3.

4.4.2 Elementi relativi alla resa grafica

Se la sincronizzazione degli elementi, che fanno parte della nostra applicazione, è resa possibile grazie al sistema operativo di Croquet, la loro resa grafica è gestita invece attraverso Babylon.js [1].

Nello specifico, le View realizzate si compongono di un elemento chiamato `SceneManager`, il quale, sulla base delle indicazioni fornite dalla View, si occupa del rendering degli elementi che devono essere mostrati all'utente, attraverso il framework Babylon.js.

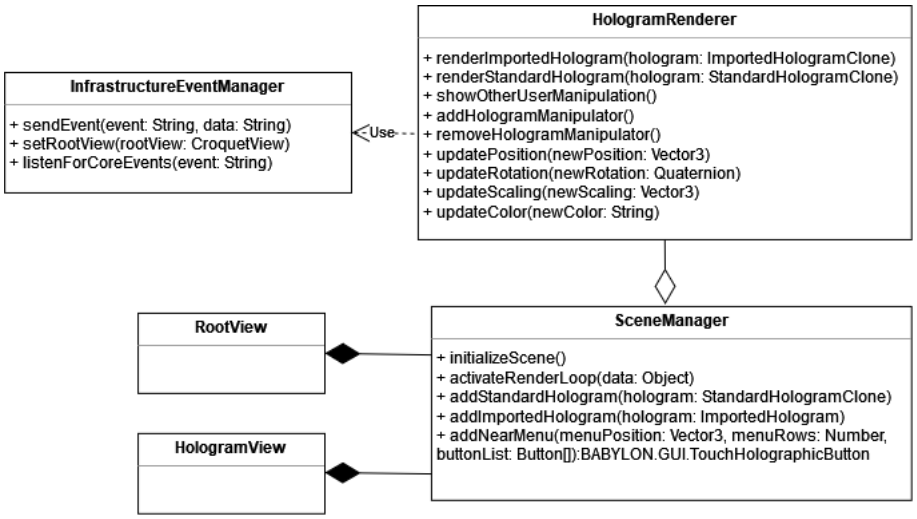


Figura 4.12: Diagramma delle classi: SceneManager e HologramRender

Come possiamo vedere dalla figura 4.12 abbiamo che per ogni ologramma che viene creato si avrà un `HologramRender`, il quale si occuperà di gestire il rendering degli elementi grafici che lo riguardano. I diversi `HologramRender`, associati ai differenti ologrammi, vengono mantenuti e gestiti attraverso la componente `SceneManager`.

4.4.3 Elementi del Model

Anche il Model del nostro sistema, come possibile vedere nella figura 4.13 si divide in più parti:

- un `RootModel`, che rappresenta il Model principale della nostra applicazione, incaricato di gestire le `RootView` che si collegano alla sessione e di istanziare i *sub-models* necessari per svolgere le operazioni richieste;

- un `HologramModel`, il quale ci rappresenta un *sub-model*, che si occupa di gestire la creazione e gli aggiornamenti degli ologrammi. In particolare, quando un utente interagendo con la scena aggiorna uno stato dell'ologramma, l'`HolgoramModel` si occupa di informare la parte Core dell'accaduto, tramite l'utilizzo di `InfrastructureEventManager`;

- un `StandardObjectModel`, il quale si occupa di gestire gli oggetti standard all'interno dell'applicazione, per far sì che ogni istanza dell'appilcativo ne visualizzi lo stesso stato, inviando gli eventi di aggiornamento sempre tramite `InfrastructureEventManager`;

- un `AnimationModel`, che ci rappresenta un sub-model istanziato dal `RootModel`, incaricato di gestire il tempo delle animazioni. Sappiamo, infatti, che all'interno di Croquet vi è un **orologio che scandisce il tempo**, il quale risulta essere uguale per tutti; un `AnimationModel`, di conseguenza, imposta il tempo che si vuole che passi per richiamare l'animazione e tramite `InfrastrucutureEventManager`, invierà l'evento indicante che il tempo è scaduto.

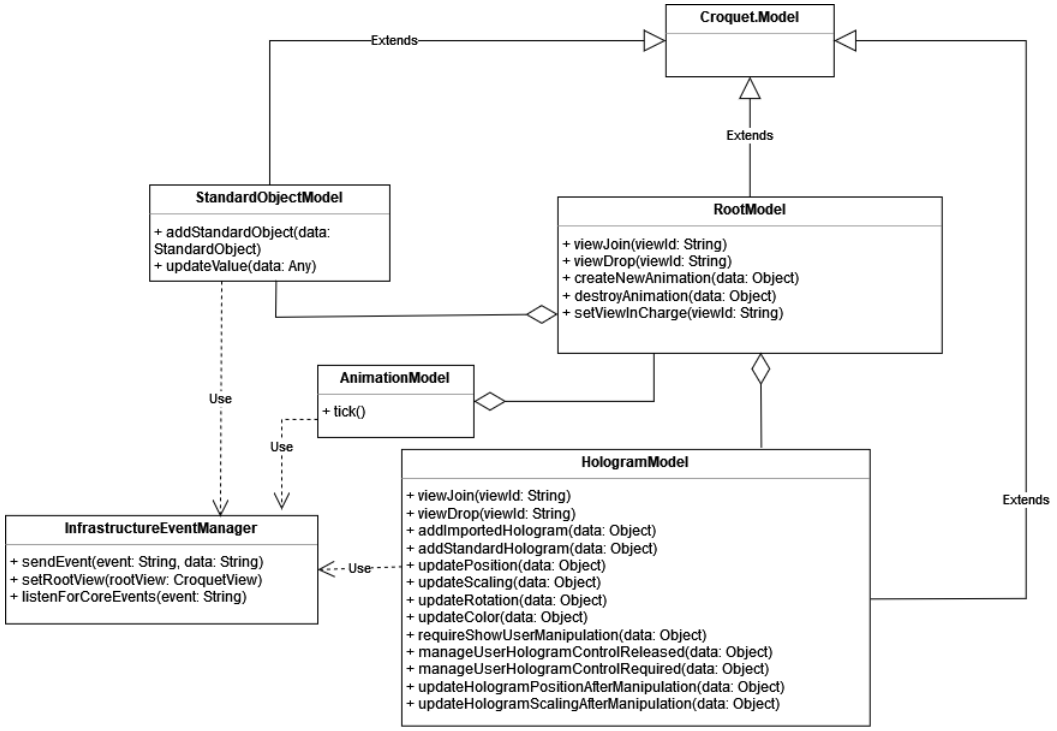


Figura 4.13: Diagramma delle classi: RootModel, HologramModel, SynchornizedVariableModel e AnimationModel

Nella parte Infrastructure, quando si aggiorna una proprietà, ad esempio di un ologramma, è l'`HologramModel` stesso che si occupa di informare la parte Core, per mezzo di `InfrastructureEventManager`, in quanto detentore di questi elementi e incaricato di aggiornarne le loro proprietà a seguito degli eventi pubblicati dalla View, al contrario di prima, nella parte Core, in cui era l'elemento stesso che si occupava di aggiornare l'altra parte, per mezzo di `CoreEventManager`. Per maggiori dettagli relativamente a questi aspetti vedere la sezione 4.5

4.5 Problematiche affrontate e scelte implementative rilevanti

All'interno di questa sezione, si andranno ad analizzare alcune problematiche di rilievo affrontate e verranno discussi dettagli implementativi ritenuti importanti, per comprendere la soluzione realizzata, cercando di spiegare, dove necessario, come mai siano state effettuate determinate scelte invece che altre.

4.5.1 Limitazioni e vincoli del Model di Croquet

Nella sezione 3.3, abbiamo parlato del framework Croquet descrivendo il suo funzionamento e le potenzialità che questi offre per poter realizzare le nostre applicazioni.

Tuttavia, riuscire ad integrare Croquet nel framework realizzato non è stato così immediato, questo perché di fatto egli richiede di strutturare la nostra applicazione in un certo modo, prevedendo due componenti principali Model e View. Nello specifico, per quanto riguarda il Model, egli **non può dipendere da altri componenti**, ma solamente gli altri elementi possono dipendere da lui. Cosa significa questo? Significa, sostanzialmente, che partire da un elemento Model di Croquet per raggiungerne un altro, non appartenente a questo dominio (Croquet) è possibile, ma non il viceversa, questo però non è valido per la View, è possibile, infatti, richiamare direttamente metodi di una View di Croquet tramite altri componenti, come accade nel nostro caso per `InfrastructureEventManager`.

La View, quindi, non impone particolari vincoli per il suo utilizzo, al contrario del Model e per chiarire questi aspetti, di seguito vengono riportati i vincoli principali nell'utilizzo degli elementi di Croquet, i quali indicano quello che **non è possibile fare**, tenendo presente che alcune di queste regole sono riportate nella documentazione di Croquet [4] e altre, invece, sono state scoperte lavorando direttamente con lo strumento stesso.

- non è possibile creare direttamente un Model tramite la *keyword* `new`, ma bisogna utilizzare il metodo `create` appositamente definito dal framework;
- un Model può essere creato solo da un altro Model;
- solamente un Model può richiamare direttamente i metodi di un altro, alcun altro componente che non sia un Model di Croquet può farlo;
- la View può interagire con i Model solo attraverso la pubblicazione di eventi;

Mettere in chiaro queste regole è molto utile, sopratutto perché il Model di Croquet è uno strumento molto importante per la sincronizzazione, infatti, gli elementi che si trovano all'interno di esso, **se opportunamente gestiti**, risulteranno essere sincronizzati fra tutti gli utenti.

A causa di questi vincoli, abbiamo messo in chiaro che **per chiedere al Model di fare qualcosa dobbiamo per forza pubblicare un evento tramite la View** e nelle prossime sezioni, potremo vedere come questa regola abbia influenzato alcuni aspetti di implementazione del framework.

4.5.2 Un unico Client al comando

Giunti a questo punto, sappiamo che a ogni nuovo utente che si connette è associato un Client, tra tutti gli utenti connessi, il primo partecipante all'esperienza (il primo Client) sarà colui **incaricato di inviare gli eventi di aggiornamento**.

La domanda ora sorge spontanea: come mai un solo Client è incaricato di inviare gli aggiornamenti e non tutti?

Quando si utilizza Croquet normalmente, per lo scopo di realizzare applicazioni sincronizzate, bisogna solo scrivere il codice della View e del Model, dopodiché non appena un nuovo utente si connette alla sessione, sarà subito in grado di vedere lo stato attuale in cui questa si trova. Nel nostro caso, invece, siccome richiamiamo le funzionalità da Croquet dall'esterno, per mezzo degli eventi inviati alla `RootView`, quello che accade è che il progettista scrivendo il programma utilizzando il framework, sulla base delle operazioni richieste, invierà degli eventi alla parte infrastrutturale contenente gli elementi di Croquet.

Nello specifico, abbiamo visto nel listato 4.1 come la prima operazione, che si deve richiedere di compiere, sia sempre la creazione di una sessione e il collegamento a questa, dopodiché possono essere specificati tutti gli elementi che vogliamo che la nostra applicazione abbia e le loro proprietà. Quando un nuovo Client viene fatto partire, tutto quello contenuto all'interno della View **non c'è** (ad esempio i menu), deve essere creato, ma tutto quello contenuto nel Model (elementi sincronizzati), se egli non è il primo utente connesso, **c'è già**, difatti se tutti i Client inviassero gli eventi di aggiornamento, si **ripeterebbero operazioni non necessarie**, perché gli elementi sincronizzati sono già presenti. Quindi, è necessario che solamente il primo Client collegato invii gli eventi di aggiornamento relativi agli elementi sincronizzati, gli altri non hanno bisogno di farlo.

Nello specifico, il Client in carica, dell'invio degli aggiornamenti, viene stabilito dal `RootModel`, il quale associa questo compito alla prima `RootView` che si collega, attraverso l'utilizzo del suo identificativo. Infatti, **l'identificativo**

delle View, è l'unico elemento che abbiamo a disposizione per distinguere i diversi Client collegati, quindi, tale elemento si presta molto bene, ad essere adottato per selezionare la View in carica.

Il `RootModel`, come mostrato nel listato 4.7, possiede due metodi, richiamati a seguito della ricezione di due eventi specifici della sessione Croquet, che li consentono di sapere quando una nuova View si è collegata e quando una View si sconnette, ottenendo il loro identificativo. Il `RootModel`, pertanto, mantiene una lista delle View collegate, la prima è quella in carica e se questa si dovesse sconnettere, il controllo verrà ceduto alla successiva e così via.

```
/**
 * Handle a new connected view.
 * @param {any} viewId the id of the new view connected.
 */
viewJoin(viewId){
    console.log(this.linkedViews);
    this.linkedViews.push(viewId);
}

/**
 * Handle the view left event.
 * @param {any} viewId the id of the outgoing view.
 */
viewDrop(viewId){
    this.linkedViews.splice(this.linkedViews.indexOf(viewId),1);

    if(viewId === this.viewInCharge){
        this.viewInCharge = this.linkedViews[0];
        console.log(this.viewInCharge)
        this.publish(this.viewInCharge, "setUpdate");
    }
}
```

Listato 4.7: Gestione della View al comando

4.5.3 Gestione della sincronizzazione degli elementi

Per quanto abbiamo detto precedentemente, le funzionalità del Model di Croquet verranno richiamate attraverso la pubblicazione degli eventi da parte della View, pertanto, ad esempio, quando l'utente richiede l'aggiunta di un ologramma, siccome vogliamo sincronizzare le sue proprietà per fare in modo che quando avviene una modifica su questi, tutti i partecipanti siano in grado di percepirla, dobbiamo inserirlo all'interno del Model. Per consentire questo

la `RootView`, a seguito della richiesta di creazione di un ologramma da parte del programmatore, pubblica un evento il cui messaggio associato contiene i dati serializzati del nostro ologramma, come si può vedere dal listato 4.8.

```
eventBus.on("createStandardHologram", (data) => {
            this.#log("create standard hologram")
            this.view.notifyEventToModel("create",
                "standardHologram", {view: this.view.viewId, hologram:
                JSON.parse(data)});
        } );
```

Listato 4.8: Pubblicazione da parte della View di un evento che richiede la creazione di un ologramma

A questo punto il `RootModel` quando riceve l'evento **deserializza** i dati contenuti nel messaggio e ricostruisce l'oggetto originale. L'oggetto così ottenuto però non è lo stesso di partenza, essi infatti si riferisce a un nuovo oggetto creato mediante l'operazione di deserializzazione, questi due oggetti sono **identici ma non uguali** e di conseguenza, all'interno del Model è presente una "copia" dell'ologramma creato. Tale copia verrà utilizzata dalla parte infrastrutturale e l'originale presente all'interno della parte Core verrà mantenuta aggiornata a seguito delle modifiche che su di questa possono avvenire.

Quindi, prendendo sempre a riferimento gli ologrammi, ne risulta che questi possono essere modificati da ambo i lati (figura 4.14); dalla parte Core a seguito delle richieste effettuate dal programmatore e dalla parte Infrastructure a seguito delle manipolazioni che l'utente su di questi può effettuare.

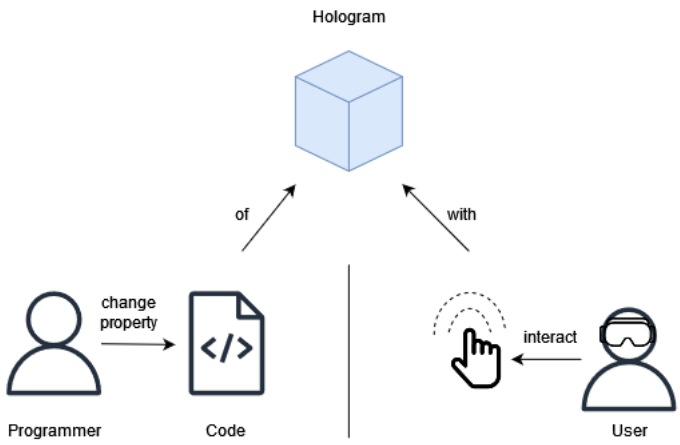


Figura 4.14: Due modi con cui le proprietà degli ologrammi possono cambiare

Gestione della sincronizzazione degli elementi dalla parte Core

Gli elementi presenti nella parte Core sono mantenuti aggiornate per mezzo del componente `SynchronizedElementUpdater`, il quale viene richiamato dall'elemento `CoreEventManager`, quando viene ricevuto un evento di aggiornamento che li compete (per maggiori dettagli riguardo agli eventi scambiati consultare la sezione 4.6).

Nella figura 4.15 possiamo notare che, quando il programmatore modifica lato codice una proprietà di un elemento, che si vuole mantenere sincronizzato, l'entità modificata notifica l'aggiornamento avvenuto a `CoreEventManager`, richiedendoli di inviare un apposito evento che notifichi l'accaduto, tale evento verrà ricevuto dalla parte Infrastructure, precisamente dalla componente `InfrastructureEventManager`, il quale si occuperà di gestirlo.

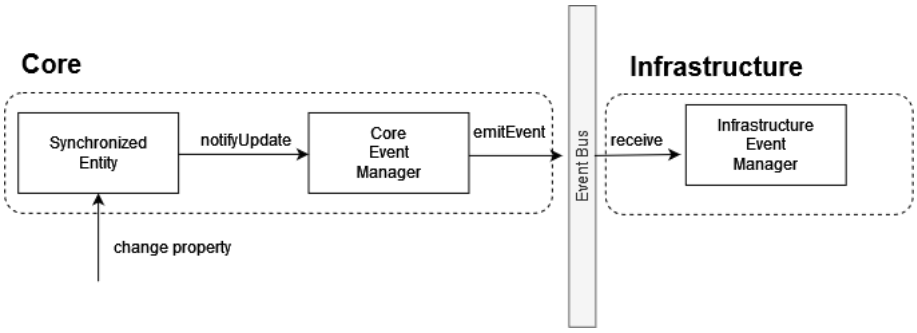


Figura 4.15: Aggiornamento elemento sincronizzato dalla parte Core

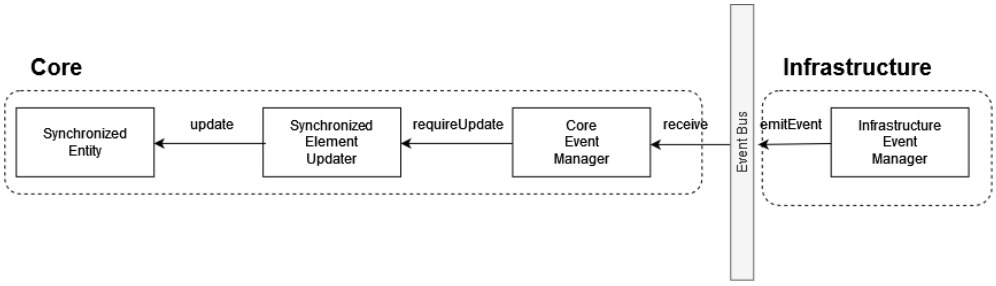


Figura 4.16: Aggiornamento elemento sincronizzato dalla parte Core

Osservando il secondo schema (figura 4.16), invece, possiamo vedere che se avviene una modifica a una copia dell'elemento sincronizzato all'interno della parte Infrastructure, sarà la componente `InfrastructureEventManager` che si occuperà di emettere un evento sull'`eventBus`, che notificherà l'accaduto, il quale verrà ricevuto da `CoreEventManager`, che richiederà a `SynchronizedElementUpdater` di eseguire l'aggiornamento richiesto.

Attraverso l'adozione di questa soluzione, abbiamo che nessun elemento all'interno della parte Core interagisce direttamente con la parte Infrastructure, garantendo indipendenza fra queste due parti e solamente l'elemento `CoreEventManager`, si occupa di inviare e ricevere gli eventi dall'`EventBus`, mantenendo **distinta** la parte di esecuzione dell'operazione e la parte di notifica di questa.

Gestione della sincronizzazione degli elementi dalla parte infrastrutturale

Completiamo gli schemi visti prima considerando anche quanto avviene nella parte infrastrutturale.

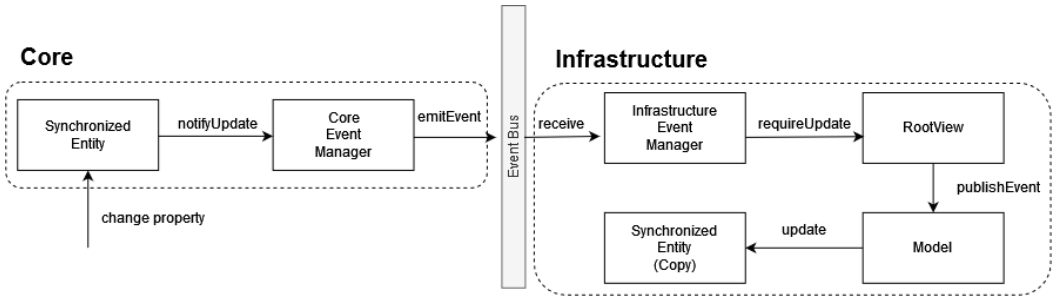


Figura 4.17: Aggiornamento elemento sincronizzato dalla parte Core schema completo

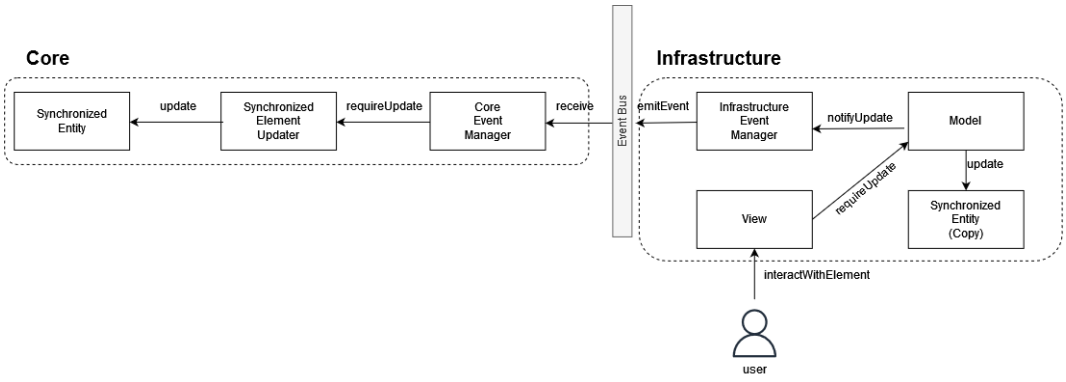


Figura 4.18: Aggiornamento elemento sincronizzato dalla parte Infrastructure schema completo

Il primo schema in figura 4.17 ci mostra che, una volta ricevuto l'evento di aggiornamento, `InfrastructureEventManager` chiederà di inviare un

messaggio alla `RootView` diretto alla componente Model, il quale informa dell'accaduto in modo tale che questi aggiorni la relativa copia presente nella parte infrastrutturale. In questo modo, tutti gli elementi della scena, per i diversi clients collegati, risulteranno essere aggiornati; ricordiamo, infatti, che gli eventi inviati dalla View verranno ricevuti da tutti i Model dei diversi utenti connessi, quindi, tutti loro avranno ricevuto il messaggio e **l'aggiornamento verrà eseguito per tutti gli utenti**, consentendoli di poter osservare lo stesso stato dell'entità modificata.

Nel secondo schema (figura 4.18), invece, possiamo osservare meglio che cosa accade quando, ad esempio, l'utente interagendo con un ologramma presente nella scena ne modifica una sua proprietà. Per prima cosa, l' `HologramView` viene notificata di questo cambiamento, richiedendo all'`HologramModel` di modificare la proprietà che l'utente vuole cambiare, fatto questo, il Model richiede a `InfrastructureEventManager` di inviare un evento che informi del cambiamento, in modo che anche l'elemento relativo alla parte Core risulti essere aggiornata.

Per spiegare come avviene l'aggiornamento degli elementi sincronizzati abbiamo preso a riferimento un'entità con cui il partecipante all'esperienza può interagire, come gli ologrammi, tuttavia, le stesse considerazioni valgono per tutti gli elementi che possono essere sincronizzati come ad esempio gli `StandardObject`, solo che in questo caso la modifica del valore di un oggetto sincronizzato nella parte infrastrutturale non è dovuta a un'interazione con l'utente, non essendo questo elemento pensato per relazionarsi con esso, ma questa verrà **effettuata direttamente dal Model** per garantire il suo corretto comportamento, i passi visti prima però rimangono comunque validi, l'unica cosa che manca è l'interazione con l'utente.

4.5.4 Utilizzo della programmazione asincrona

Analizzando quanto abbiamo detto fino ad ora del funzionamento del framework, possiamo notare come si sia fatto uso della programmazione basata sugli eventi e della programmazione asincrona.

Osservando i listati 4.1 e 4.2, possiamo vedere come alcuni metodi di `SessionManager` e `Scene` restituiscano al programmatore una `Promise`. In particolare, i metodi che fanno questo sono tre: uno in `SessionManager` (listato 4.9), per la creazione della sessione e due in `Scene` per l'aggiunta degli ologrammi (listato 4.10).

```
/**
* Async method that start a session.
* @param apiKey {String} the api key.
* @param appId {String} the app id.
```

```
 * @returns {Promise<boolean>}
 */
async startSession(apiKey, appId){
    if (typeof apiKey === undefined || typeof appId === undefined){
        throw new Error("parameters apiKey and appId can't be
            undefined!");
    }

    const croquetSession = new CroquetSession();
    await croquetSession.sessionJoin(apiKey, appId);

    return new Promise((resolve) =>
    {
        this.#log("SessionManager: session started true");
        this._sessionStarted = true;
        resolve(true);
    });
}
```

Listato 4.9: metodo startSession di SessionManager

```
/**
 * Add a new importedHologram to the scene.
 * @param hologram {ImportedHologram} the new hologram to add.
 * @returns {Promise<boolean>} representing the insertion made.
 */
addImportedHologram(hologram){
    this.#verifyIfSceneIsInitialized();
    this.#verifyIfElementNotExist(hologram.name);
    synchronizedElementUpdater.addHologram(hologram);
    coreEventManager.sendEvent("createImportedHologram",
        JSON.stringify(hologram));

    return new Promise((resolve) => {
        coreEventManager.listenForInfrastructureEvent(
          "importedHologramCreated", () => {
            resolve(true)
        });
    });

}

/**
 * Add a new StandardHologram to the scene
```

```
 * @param hologram {StandardHologram} the new hologram to add.
 * @returns {Promise<boolean>} representing the insertion made.
 */
addStandardHologram(hologram){
    this.#verifyIfSceneIsInitialized();
    this.#verifyIfElementNotExist(hologram.name);
    synchronizedElementUpdater.addHologram(hologram);
    coreEventManager.sendEvent("createStandardHologram",
        JSON.stringify(hologram));

    return new Promise((resolve) => {
        coreEventManager.listenForInfrastructureEvent(
          "standardHologramCreated", () => {
            resolve(true)
        });
    });
}
```

Listato 4.10: metodi addImportedHologram e addStandardHologram di Scene

La programmazione asincrona, in questo caso, viene utilizzata in quanto le operazioni demandate richiedono del tempo per poter essere realizzate. Una volta che i componenti della parte Core (`SessionManager` e `Scene`) inviano la richiesta di ciò che deve essere eseguito dalla parte Infrastructure, tramite `CoreEventManager`, si mettono in attesa dell'evento, che li informi che l'operazione è avvenuta; dopo aver ricevuto tale evento, la `Promise` restituita al programmatore potrà essere risolta e sarà possibile procedere con le altre operazioni.

4.6 Interazione fra i componenti del sistema

Analizziamo maggiormente in dettaglio quali siano le interazioni che avvengono fra i componenti del sistema.

Consideriamo di andare a modificare le proprietà di un ologramma, verificando prima, come interagiscono fra loro gli elementi della parte Core e poi come interagiscono fra loro gli elementi della parte Infrastructure.

Come possiamo vedere dalla figura 4.19, l'elemento `StandardHologram`, a seguito della richiesta di cambiamento di colore, verifica se il Client su cui è in esecuzione sia quello in carica di mandare gli aggiornamenti, attraverso il componente `SynchronizedElementManager`, in caso affermativo cambia la proprietà del colore e richiede a `CoreEventManager`, di inviare un evento sull'`EventBus` che informi dell'accaduto.

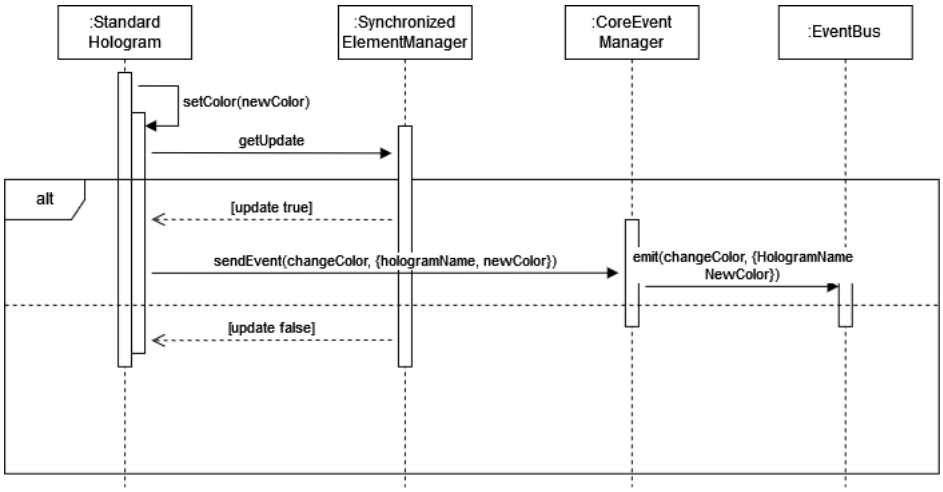


Figura 4.19: Diagramma di sequenza: interazione fra le entità della parte Core per la modifica del colore di un ologramma

A questo punto l'evento è giunto nella parte infrastrutturale e verrà gestito da `InfrastructureEventManager`, il quale come illustrato nella figura 4.20, richiederà alla `RootView` di notificare la componente Model, del cambiamento di colore dell'ologramma. L'`HologramModel`, incaricato di gestire questo tipo di eventi, si occuperà di modificare la proprietà della sua copia, che fa riferimento all'ologramma modificato in Core, in modo da avere una versione aggiornata.

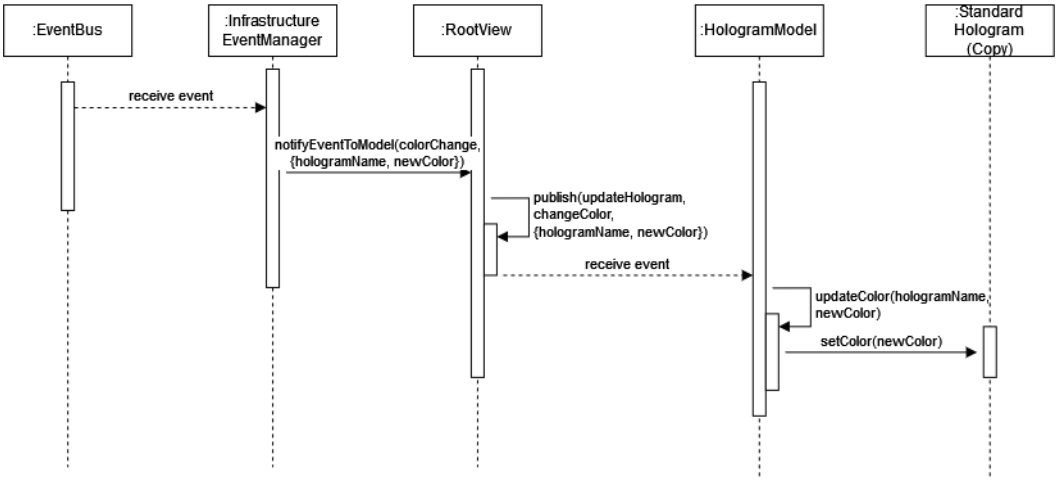


Figura 4.20: Diagramma di sequenza: interazione fra le entità della parte Infrastructure per la modifica del colore di un ologramma (parte Model)

Giunti fino a qui, abbiamo ottenuto l'aggiornamento della proprietà dell'ologramma, tuttavia, siccome abbiamo detto che ad essi è associata anche

un parte grafica, occorre aggiornare anche quella rappresentazione, sulla base dell'operazione che è stata svolta.

Come possiamo vedere dal diagramma delle interazioni sottostante, (figura 4.21), l'`HologramModel` si occuperà di pubblicare un evento che informa la View del cambiamento avvenuto sull'ologramma, l'`HologramView` riceverà tale evento e richiederà all'`HologramRender` associato all'ologramma, ottenuto tramite `SceneManager`, di modificarne il colore.

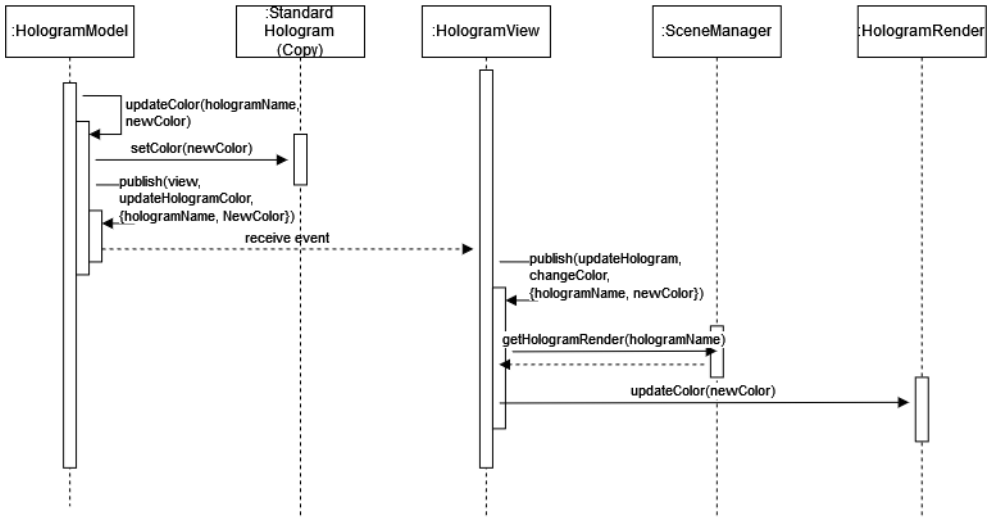


Figura 4.21: Diagramma di sequenza: interazione fra le entità della parte Infrastructure per la modifica del colore di un ologramma (parte View)

Quest'ulteriore passaggio, mostrato dal diagramma di interazione in figura 4.21, è necessario affinché tutti i partecipanti all'esperienza vedano la modifica, infatti, ragionando in termini di quello che viene svolto dal sistema operativo di Croquet abbiamo che: **una sola View** riceverà l'evento di modifica del colore, quella in carica, tale View pubblicherà un evento che viene ricevuto da tutti i Model collegati, che aggiorneranno tutti la loro copia dell'ologramma e poi richiederanno alle rispettive View, tramite l'invio di un **evento in locale**, di mostrare la modifica avvenuta.

Analizziamo adesso il caso in cui sia il partecipante all'esperienza, che interagendo con l'ologramma presente nella scena, ne modifichi la sua posizione, grazie alle funzionalità offerte dal menu di manipolazione, descritto precedentemente.

In questo caso, la modifica parte direttamente dalla View, quindi, l'ordine degli eventi è inverso rispetto a prima, ci muoveremo dalla parete Infrastructure verso la parte Core.

Osservando il diagramma delle interazioni in figura 4.22 possiamo vedere che, la View su cui si è verificata l'interazione (dove è stata modificata la proprietà dell'ologramma) pubblicherà un evento ricevuto dall'`HologramModel`, il quale aggiornerà le proprietà dell'ologramma manipolato e dopodiché invierà un messaggio a tutte le altre View collegate, esclusa quella su cui è avvenuta l'interazione (in quanto non necessita di essere aggiornata), richiedendoli di modificare la posizione dell'ologramma a seguito della manipolazione avvenuta. Tale aggiornamento si effettua in modo identico a come visto prima per il colore (nel diagramma in figura figura 4.20.), solo che questa volta la proprietà da modificare è la posizione.

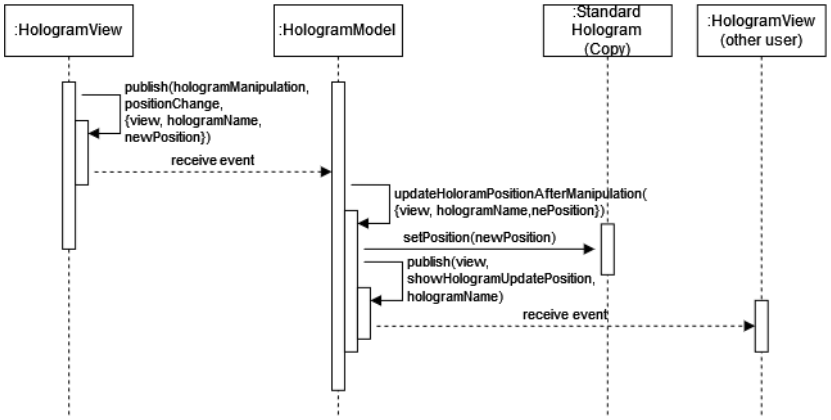


Figura 4.22: Diagramma di sequenza: interazione fra le entità della parte Infrastructure per la modifica della posizione di un ologramma

Giunti a questo punto, la View è aggiornata e tutti gli utenti vedono l'ologramma nella posizione corretta, tuttavia, l'elemento nella parte Core non è ancora stato modificato e se non si cambiano anche le sue proprietà, si rischia che in seguito le modifiche che verranno effettuate siano inconsistenti.

Per poter risolvere questo problema, l'`HologramModel`, tramite `InfrastructureEventManager`, invierà l'evento di aggiornamento sull' `EventBus` che verrà ricevuto dalla parte Core, come illustrato nel diagramma d'interazione in figura 4.23.

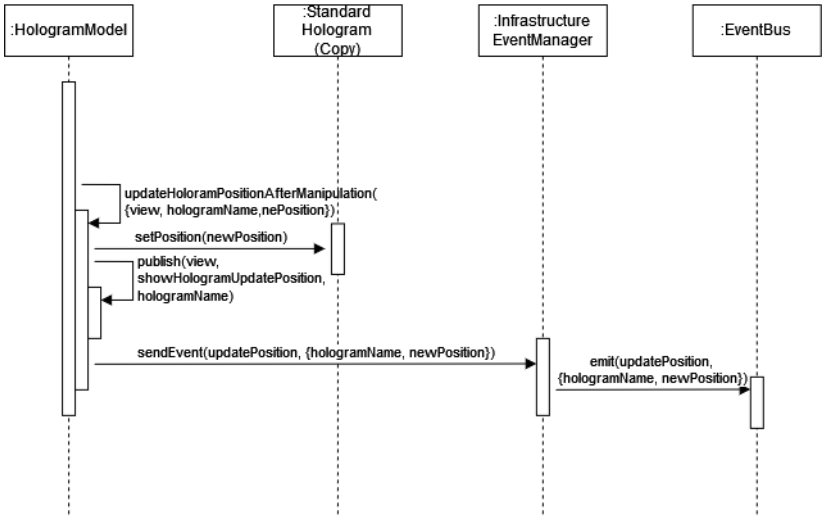


Figura 4.23: Diagramma di sequenza: interazione fra le entità della parte Infrastructure per la modifica della posizione di un ologramma

Quando l'evento verrà ricevuto da `CoreEventManager`, egli si occuperà di richiedere a `SynchornizedElementUpdater`, l'aggiornamento della proprietà opportuna dell'ologramma specificato (figura 4.24). In questo modo anche l'ologramma della parte Core risulterà essere aggiornato, rispetto all'interazione che si ha avuto con l'utente, garantendo così la sua consistenza.

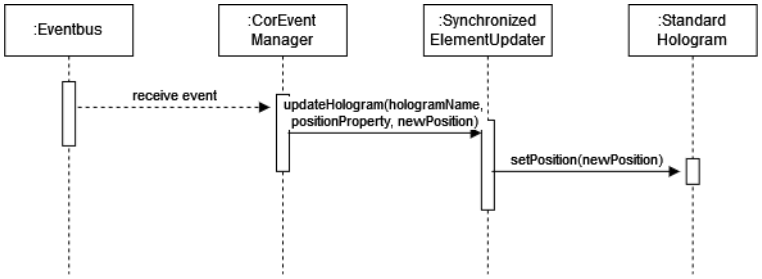


Figura 4.24: Diagramma di sequenza: interazione fra le entità della parte Infrastructure per la modifica della posizione di un ologramma

Capitolo 5

Considerazioni finali e sviluppi futuri

Giunti a questo punto abbiamo terminato l'analisi sul framework realizzato, il quale ci fornisce uno strumento con cui poter andare a realizzare esperienze di eXtended Reality collaborative; questo però non significa che tale strumento sia completo, anzi le funzionalità offerte oltre che ad essere solo l'inizio di ulteriori capacità che potranno essere inserite, possono sempre essere migliorate in virtù di nuove considerazioni, nuovi sviluppi e innovazioni in ambito tecnologico.

In questo capitolo andiamo a considerare il framework nel suo complesso, andando a tirare le somme di quello che è stato fatto valutando la soddisfazione dei requisiti individuati, dei vantaggi e degli svantaggi che questa soluzione offre e di quali siano, eventualmente, aspetti e funzionalità che possano essere considerati o aggiunti in un futuro.

5.1 Soddisfacimento dei requisiti

Ci chiediamo se il framework realizzato è in grado di soddisfare i requisiti: funzionali, non funzionali, utente e di implementazione visti nella sezione 4.1. Iniziando da quelli funzionali, possiamo riscontrare che la soluzione adottata ci consente di rispettare i requisiti individuati, tramite di essa è effettivamente possibile svolgere le funzionalità richieste, quali: creare un mondo aumentato in cui poter inserire degli elementi, gestire l'interazione con gli utenti mediante apposite interfacce grafiche etc.

Per quanto riguarda i requisiti non funzionali, invece, possiamo dire che anche questi sono stati rispettati, in particolare, il rispetto dei primi due (realizzare una soluzione modulare e in grado di scalare a seguito dei cambiamenti) ci consente di gestire la complessità del nostro sistema e per poterli assolvere, sono stati tenuti in considerazione due importanti principi di qualità del

software: il *Single Responsability Principle*, il quale ci dice che, un componente software dovrebbe avere una e solo una, ragione per cambiare e il *Keep it Simple Stupid*, che ci consiglia di preferire sempre soluzioni semplici invece che complesse dove possibile, in quanto, la maggior parte dei sistemi hanno la possibilità di funzionare meglio quando vengono mantenuti semplici [39].

Nello sviluppo dell'architettura per il framework, si è cercato di rispettare quanto più possibile questi due principi, si ritiene, infatti, che ogni componente software abbia un compito ben stabilito e svolga solamente le funzionalità che li competono, questo fa sì, quindi, che se si rende necessario aggiungere un componente o modificarne una certa funzionalità fornita, il minor numero di elementi possibile vengano cambiati.

Anche per quanto riguarda l'ultimo requisito non funzionale, si è cercato di andarlo a soddisfare quanto più possibile, tenendo conto che non è mai facile lavorare con un sistema distribuito, si ritiene comunque di essere giunti a una soluzione di *eventually consistence*, il che significa, che alla fine l'applicazione realizzata, i dati presenti in essa, le proprietà degli ologrammi e degli oggetti standard, risulteranno essere consistenti e opportunamente gestiti.

Avendo avuto modo di utilizzare il framework per la realizzazione di alcune semplici applicazioni, alcune delle quali possono essere viste nell'appendice A, si può dire che anche i requisiti utente sono stati soddisfatti. La soluzione adottata ci fornisce un modo, abbastanza semplice, con cui poter costruire applicazioni di eXtended Reality collaborative e i suoi componenti volendo possono essere utilizzati dal programmatore anche per costruire nuovi elementi, in relazione allo scopo che egli vuole raggiungere.

Infine, possiamo dire che il framework realizzato rispetti i requisiti implementativi enunciati in precedenza; egli è in grado di realizzare delle applicazioni scritte in linguaggio Javascript che supportino WebXR e funzionino sui principali browser di riferimento.

5.2 Molti eventi da gestire

Una critica che si può muovere nei confronti della soluzione adottata è che, per avere effettivamente un aggiornamento degli elementi, sopratutto se questo riguarda un elemento sincronizzato, diversi passaggi devono essere eseguiti e diversi eventi devono essere scambiati. Questo è dovuto, in parte, alle regole imposte da Croquet per garantire la sincronizzazione degli elementi e in parte, invece, alla complessità della realizzazione di soluzioni di questo tipo, in cui ricordiamo, abbiamo a che fare con un sistema distribuito e vogliamo garantire a tutti i partecipanti di poter vivere un'unica esperienza condivisa. Ciò non toglie però che in un futuro questo aspetto non possa essere migliorato.

Infine, un ulteriore svantaggio che si può riscontrare, riguarda il fatto che sebbene l'aggiunta di un nuovo elemento può non richiedere un cambiamento degli elementi precedenti, potrebbe allo stesso modo, domandare l'aggiunta di ulteriori eventi che debbono essere scambiati per garantire le funzionalità richieste, i quali si andrebbero a sommare a quelli già presenti nel frameowrk e dovranno essere gestiti opportunamente.

5.3 Resistenza ai cambiamenti

Nell'ambito dell'eXtended Reality, essendo questo un ambito nato di recente, vi sono continui sviluppi sia in campo teorico, in relazione allo studio che viene fatto di questi sistemi, che nel pratico, in relazione a nuovi strumenti e tecnologie che nascono per fornire supporto alla realizzazione di sistemi di questo tipo.

Le cose, pertanto, tendono a cambiare abbastanza in fretta, si pensi ad esempio che Unity [36], uno dei principali strumenti software per poter realizzare contenuto 3D da mostrare all'utente, viene aggiornato con una cadenza quasi mensile, portando cambiamenti anche radicali nei servizi offerti, i quali rischiano di far sì che applicazioni realizzate il mese precedente non siano più compatibili con la nuova versione. Anche il supporto fornito a WebXR [31] può cambiare da un giorno all'altro, a causa degli aggiornamenti del Browser, ad esempio ad aprile 2023, HoloLens poteva accedere tramite WebXR a sessioni sia di Augmented che di Virtual Reality attraverso il browser Edge, poi a seguito dell'aggiornamento di maggio del motore di ricerca, ora è possibile accedere solamente a sessioni di Virtual Reality.

Questi esempi riportati ci mostrano come le applicazioni che andiamo a realizzare siano dipendenti dalle decisioni e modifiche effettuate dai provider degli strumenti adottati. Pertanto, la velocità con cui avvengono i cambiamenti, per la realizzazione del framework, ha fatto insorgere la necessità di realizzare la soluzione in modo tale che, questa fosse quanto più possibile indipendente rispetto alle tecnologie adottate, cosicché se in un futuro venissero fatti degli aggiornamenti o nuovi strumenti dovessero nascere, questi possano essere facilmente integrati o utilizzati per sostituire quelli precedenti.

Difatti, possiamo notare come la parte Core del framework, con cui lo sviluppatore interagisce direttamente, possa rimanere immutata nel tempo, mentre la parte Infrastructure, coinvolgendo essa gli elementi infrastrutturali e tecnologici, sia quella che possa subire le maggiori modifiche, ma grazie al fatto che le tecnologie adottate sono state separate tramite la modellazione di concetti differenti, risulta essere semplice poter andare ad agire in modo mirato su determinati aspetti. Ad esempio, se voglio cambiare la tecnologia

di rendering passando da Babylon.js a Three.js basterà modificare solo gli elementi che utilizzano Babylon, quindi, quelli relativi alla scena e nient'altro.

5.4 Maggiore percezione dell'altro e supporto al lavoro cooperativo

Di molto si è parlato nel capitolo 2 di quanto sia importante riuscire a percepire il punto di vista dell'altro e la sua presenza nel mondo virtuale, la quale può essere ottenuta attraverso l'utilizzo di avatar che ci possono mostrare l'altra persona per intero o di altri indicatori quali lo sguardo o il movimento delle mani, che ci consentono, invece, di concentrarci su dettagli di rilievo per la comunicazione che si può andare a svolgere.

Si può pensare di andare a supportare, tramite il framework, la vista dell'altro considerando queste diverse strategie che possono essere realizzate, consentendo in un contesto di partecipazione in remoto, in cui non si ha la possibilità di poter vedere l'altro nel proprio spazio fisico, di poterne avere una sua rappresentazione virtuale veritiera, in termini di locazione orientazione e scala.

Inoltre, attualmente, il framework dà la possibilità di poter vedere e manipolare gli ologrammi, consentendo di apportare cambiamenti nella loro posizione e scala, non dà la possibilità, tuttavia, di scrivere note o fare disegni per meglio comprendere, ricordare o rimarcare quello che si sta facendo, supportando maggiormente il lavoro che si sta svolgendo. Per poter favorire la comunicazione e l'attività collaborativa dai partecipanti, pertanto, si può pensare di introdurre una lavagna condivisa (e.g [17]), o comunque un sistema di scrittura e disegno, che dia la possibilità agli utenti di annotare o evidenziare caratteristiche importanti di quello che si sta vedendo, potendo vedere anche le annotazioni lasciate dagli altri.

Infine, sempre per favorire l'interazione, si può pensare di aumentare gli elementi dell'interfaccia grafica disponibili, fornendo altri tipi di menu o oggetti con cui potersi relazionare che diano risultati prestabiliti (come per il `ManipulatorMenu`) o determinati dall'utente, consentendo di avere un certo grado di personalizzazione dell'esperienza.

Conclusioni

L'obiettivo di questa tesi era di riuscire a realizzare un framework che consentisse la realizzazione di applicazioni di eXtended Reality collaborative.

Per poter fare questo, si è partiti individuando che cosa significhi per noi, in un contesto di eXtended Reality, collaborare e quali siano i principali aspetti e sfide che nella realizzazione di questi sistemi debbano essere considerate e affrontate.

In particolare, le sfide maggiori che si pongono al programmatore, che si addentri nella realizzazione di questo tipo di applicazioni, sono consentire la percezione dell'altro e la creazione di uno spazio di riferimento condiviso.

Quando lavoriamo in gruppo, infatti, riuscire a percepire che cosa l'altro sta guardando o dove si sta concentrando la sua attenzione ci aiuta a comprendere meglio il suo punto di vista. La creazione di uno spazio di riferimento condiviso, invece, ci consente di avere la stessa visone del mondo aumentato e vedere le stesse cose che l'altro è in grado di vedere, quindi, se un partecipante sposta un ologramma verso destra, tutti saranno in grado di percepire questo spostamento, mantenendo il loro personale punto di vista sulla scena.

Riuscire a creare questo tipo di sistemi però, affrontando le sfide descritte, non è del tutto semplice, tuttavia, sono presenti strumenti e tecnologie, che ci consentono di agevolare l'implementazione di determinati aspetti delle applicazioni che vogliamo andare a creare.

WebXR [31] ci dà la possibilità di poter supportare diversi dispositivi per le diverse realtà, consentendoci di fruire i contenuti della nostra applicazione direttamente attraverso una pagina Web; tecnologie di rendering quali Babylon.js [1] e Three.js [37], ci consentono di ottenere e renderizzare la rappresentazione grafica del contenuto che vogliamo mostrare e infine, tramite Croquet [4] possiamo gestire i diversi utenti che desiderano partecipare all'esperienza, controllando gli aspetti di sincronizzazione legati agli elementi olografici, che desideriamo far vedere.

Attraverso il framework realizzato è stato possibile dimostrare come si possa integrare fra loro questi diversi strumenti per poterne creare un nuovo, che renda la realizzazione di applicazioni di eXtended Reality collaborative, meno complessa di quanto non lo sia già attualmente.

Questo però è solo l'inizio, una piccola visione di un panorama che è destinato ad espandersi. In futuro gli strumenti che ci consentiranno di costruire le nostre applicazioni, come già accaduto, aumenteranno sempre di più e auspicabilmente, le funzionalità che ci consentiranno di ottenere miglioreranno di volta in volta. Possiamo dire, infatti, che il focus dei ricercatori si potrà spostare dallo ricercare e sviluppare tecnologie che ci consentano di risolvere e affrontare le sfide per la realizzazione di questi sistemi, a concentrarsi sulle sfumature del sostegno alla collaborazione. Non solo potremo essere in grado di supportare la collaborazione attraverso i nostri sistemi, ma potremo anche andare a sviluppare nuove idee di lavoro collaborativo che prima non esistevano [8].

Questa prima versione del framework creata, negli anni a venire, potrà essere migliorata e aggiornata, consentendoci sempre di più e sempre meglio, di fornire un supporto alla collaborazione e al lavoro di gruppo tramite l'eXtended Reality e come dicono Barret Ens, Joel Lanir & Co. nel loro articolo "*Revisitin Collaboration through Mixed Reality: The Evolution of Groupware*":

> "Our work is just a starting point and more work must be invested in revising frameworks of collaboration to help describe, categorize and identify new opportunities for technology that expand our sense of what it means to be together."

Appendice A

Esempi di utilizzo

Avendo visto le caratteristihce principali del framework e la sua struttura, passiamo ora ad illustrare alcuni esempi del suo utilizzo. In particolare, in questa appendice verranno illustrati due esempi, il primo, mostrerà una semplice animazione su un ologramma standard, il secondo, invece, prevederà l'utilizzo di ologrammi importati abilitandone la loro manipolazione.

A.1 Primo esempio: una semplice animazione

Per prima cosa, come abbiamo illustrato nel capitolo 4, inizializziamo la scena ottenendo un'`apiKey` e un'`appId`, dal portale ufficiale di Croquet, fatto questo, possiamo passare a definire il nostro ologramma e le sue proprietà; nello specifico, l'animazione che vogliamo andare a realizzare, prevede di far cambiare colore all'ologramma ogni secondo. Come forma di riferimento prendiamo una sfera assegnandoli come colore di partenza il bianco, dopodiché aggiungiamo l'ologramma creato alla scena.

Il seguente listato A.1 mostra il codice scritto per svolgere gli steps descritti in precedenza e la figura A.1 mostra il risultato ottenuto fino ad ora.

```
import * as CollaborativeXR from "../../collaborativeXR.min.js"

const apiKey = 'myApiKey';
const appId = 'myAppId';

const sessionManager = new CollaborativeXR.SessionManager();
await sessionManager.startSession(apiKey, appId);
const scene = new CollaborativeXR.Scene(sessionManager);
scene.initializeScene();

const positionSphere = new CollaborativeXR.Vector3(0, 1, 1.2);
```

```
const rotationSphere = new CollaborativeXR.Quaternion(0,0,0,0);
const creationOptionsSphere = {diameter: 0.5}
const standardColor = "#ffffff";
const sphere = new CollaborativeXR.StandardHologram("Sphere",
    CollaborativeXR.StandardShape.Sphere, creationOptionsSphere,
        positionSphere,
    rotationSphere, standardColor);
await scene.addStandardHologram(sphere);

scene.activateRenderLoop();
```

Listato A.1: Semplice animazione: creazione dell'ologramma

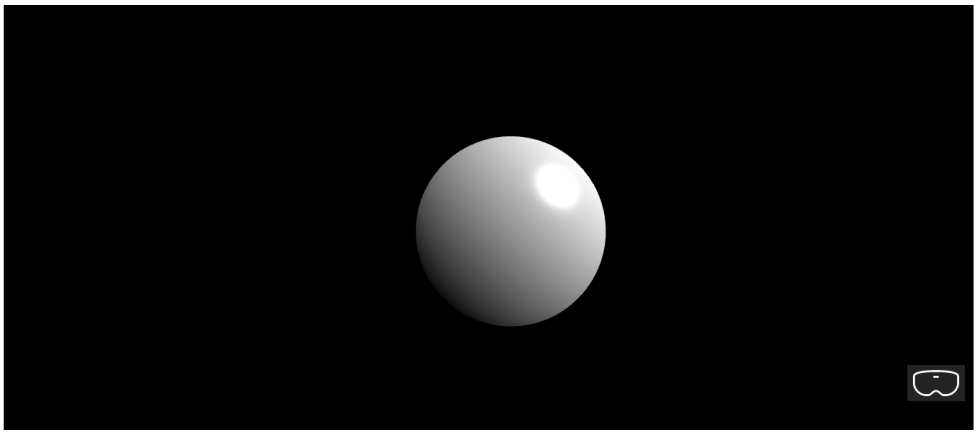


Figura A.1: Visualizzazione dell'ologramma creato

A questo punto, possiamo aggiungere l'animazione. Quello che vogliamo ottenere come risultato è che la sfera cambi di colore, passando dal rosso al blu, in base al tempo trascorso. Per tenere conto dei secondi passati utilizziamo un oggetto standard, in modo tale che, quando il valore da lui detenuto assume un valore pari, il colore della sfera sia rosso e quando assume un valore dispari, il colore della sfera sia blu.

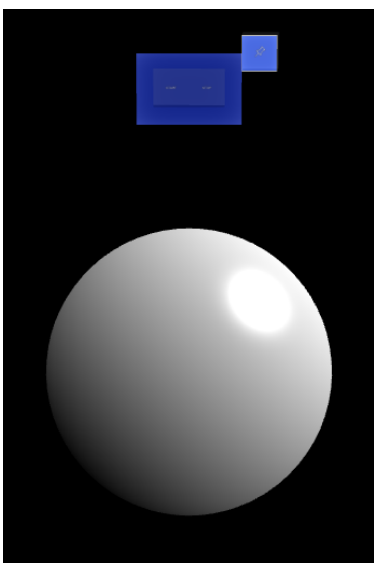
Infine, per far partire e fermare l'animazione, in modo controllato, utilizziamo un `NearMenu` costituito di due pulsanti, chiamati rispettivamente: `buttonStart`, incaricato di far partire l'animazione e `buttonStop`, incaricato di fermare l'animazione, come illustrato nel seguente listato A.2.

```
//...
const animationTime1 = 1000;
const changeColor = new CollaborativeXR.Animation("changeColor",
    animationTime1);
```

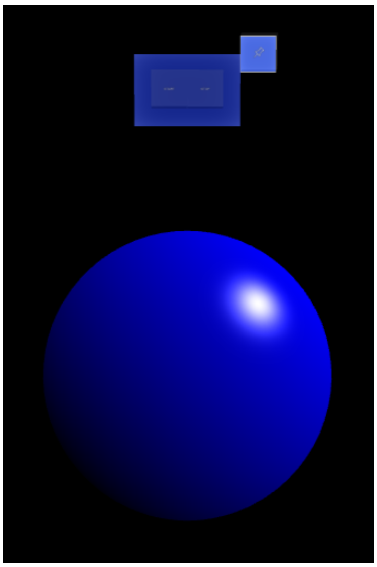
```
const counter = new CollaborativeXR.StandardObject("counter", 0);
sessionManager.addStandardObjeect(counter);

changeColor.setAnimationCallback(() => {
  counter.value = counter.value + 1;
  if(counter.value % 2 === 0){
    sphere.color = #00ff00
  }else{
     sphere.color = #0000ff
  }
});

const nRowsMenu = 1;
const nearMenuPosition = new CollaborativeXR.Vector3(0, 1.4, 0.5);
const nearMenu = new CollaborativeXR.NearMenu(nearMenuPosition,
    nRowsMenu);

const buttonStart = new CollaborativeXR.Button("buttonStart",
    "START");
buttonStart.setOnPointerDownCallback(() => {
  changeColor.startAnimation();
});
nearMenu.addButton(buttonStart);

const buttonStop = new CollaborativeXR.Button("buttonStop", "STOP");
buttonStop.setOnPointerDownCallback(() => {
  changeColor.stopAnimation();
});
nearMenu.addButton(buttonStop);

scene.addNearMenu(nearMenu);

scene.activateRenderLoop();
```

Listato A.2: Semplice animazione: aggiunta dell'animazione e di un menu

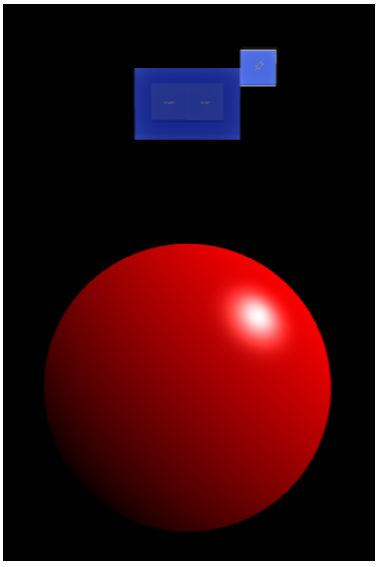
La seguente figura A.2 ci mostra il comportamento dell'applicazione, una volta premuto sul pulsante `buttonStart` il colore della sfera cambierà con lo scorrere del tempo, come volevamo.



(a) Scena iniziale



(b) Scena dopo un secondo



(c) Scena dopo due secondi

Figura A.2: Comportamento dell'applicazione

Grazie all'utilizzo dello `StandardObject` abbiamo che tutti i diversi client vedranno, nello stesso momento, il medesimo stato della scena e questi risulteranno sincronizzati fra loro, la figura A.3 ci mostra proprio questo, in cui considerando di accedere all'applicazione da due clients differenti, in questo caso di Virtual Reality, il tempo percepito è lo stesso e pure l'aggiornamento che viene eseguito sull'ologramma è il medesimo.

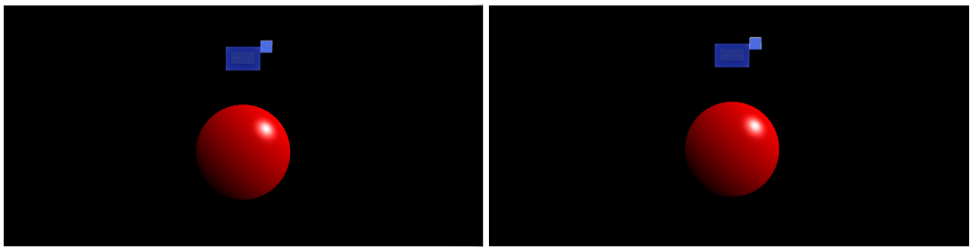


Figura A.3: Visualizzazione dell'ologramma creato

A.2 Secondo esempio: manipolazione degli ologrammi

In questo secondo esempio, vogliamo mostrare come sia possibile importare degli ologrammi, la cui resa grafica è già stata definita per mezzo di un file in formato `.gltf` e come possa essere consentita la loro manipolazione.

I modelli che decidiamo di importare sono due, rappresentanti rispettivamente due modelli di robot.

Il seguente listato A.3 mostra come creare l'applicazione desiderata. Per prima cosa inizializziamo sempre la scena, dopodiché passiamo alla creazione degli `ImportedHologram` stabilendo le loro proprietà, gli aggiungiamo alla scena e infine, associamo ad ognuno di essi un `ManipulatorMenu` per poterli modificare.

```
import * as CollaborativeXR from "../../collaborativeXR.min.js"

const apiKey = 'myApiKey';
const appId = 'myAppId';

const sessionManager = new CollaborativeXR.SessionManager();
await sessionManager.startSession(apiKey, appId);
const scene = new CollaborativeXR.Scene(sessionManager);
scene.initializeScene();

const animatedRobotRotation = new CollaborativeXR.Quaternion(0, 1,
    0, 0);
const animatedRobotScaling = new CollaborativeXR.Vector3(0.005,
    0.005, 0.005);
const animatedRobotPosition = new CollaborativeXR.Vector3(0.2, 0.2,
    1.2);
const robot1 = new CollaborativeXR.ImportedHologram("robot1",
    "./importedMesh/animated_robot/scene.gltf",
```

```
    animatedRobotPosition, animatedRobotRotation,
    animatedRobotScaling);
await scene.addImportedHologram(robot1);

const animatedRobotManipulatorPosition = new
    CollaborativeXR.Vector3(0.5, 0.8, 1.2)
const animatedRobotManipulatorMenu = new
    CollaborativeXR.ManipulatorMenu(animatedRobotManipulatorPosition,
      "robot1");
scene.addManipulatorMenu(animatedRobotManipulatorMenu);

const yellowRobotRotation = new CollaborativeXR.Quaternion(0, 1, 0,
    0);
const yellowRobotScaling = new CollaborativeXR.Vector3(0.5, 0.5,
    0.5);
const yellowRobotPosition = new CollaborativeXR.Vector3(-0.6, 1,
    1.2);
const robot2 = new CollaborativeXR.ImportedHologram("robot2",
    "./importedMesh/yellow_robot/scene.gltf", yellowRobotPosition,
    yellowRobotRotation, yellowRobotScaling);
await scene.addImportedHologram(robot2);

const blueRobotManipulatorPosition = new CollaborativeXR.Vector3(-1,
    1, 1.2)
const blueRobotManipulatorMenu = new
    CollaborativeXR.ManipulatorMenu(blueRobotManipulatorPosition,
      "robot2");
scene.addManipulatorMenu(blueRobotManipulatorMenu);

scene.activateRenderLoop();
```

Listato A.3: Manipolazione degli ologrammi: creazione degli ologrammi e aggiunta del ManipulatorMenu

A questo punto, passiamo ad eseguire il *render-loop* e a lanciare la nostra applicazione, visualizzando l'effetto prodotto mostrato in figura A.4.

Figura A.4: Visualizzazione della scena creata

Tutti gli utenti connessi vedranno la stessa rappresentazione e avranno la possibilità di manipolare i due ologrammi importati, potendo vedere le modifiche effettuate dagli altri partecipanti.

Nell'esempio mostrato in figura A.5 l'utente di destra, che ha acceduto all'applicazione attraverso un dispositivo di Augmented Reality, decide di manipolare l'ologramma di sinistra, prendendo il controllo di tale ologramma. L'altro utente potrà osservare le manipolazioni svolte senza però avere la possibilità, a sua volta, di prendere il controllo di quest'ologramma fino a quando l'utente di destra non smettera di manipolarlo.

L'interfaccia grafica adottata illustra proprio quest'ultimo passaggio descritto, il bottone del `ManipulatorMenu` dell'utente di destra ha assunto un colore verde, indicandoli che è in controllo di quell'ologramma, mentre quello dell'utente di sinistra ha assunto colore rosso, indicandoli che non potrà prendere il controllo di quest'ultimo, in quanto già manipolato da un altro utente.

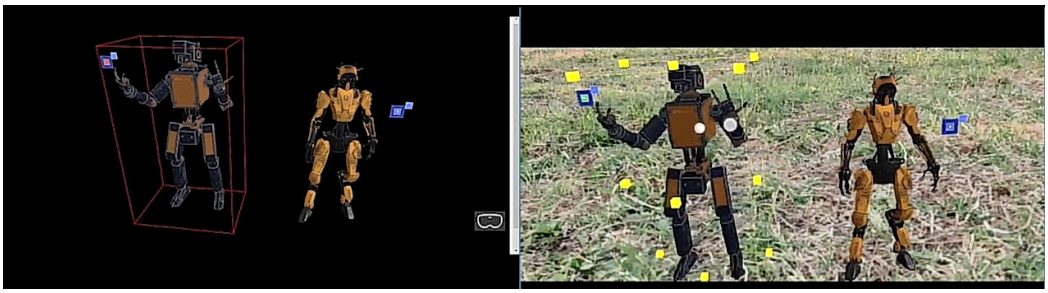


Figura A.5: Manipolazione dell'ologramma e relativa visualizzazione tramite la realtà virtuale e aumentata

Ringraziamenti

La prima persona che mi sento in dovere di ringraziare, per la realizzazione di questa tesi, è il mio correlatore Samuele Burattini, che mi ha seguito, veramente, dall'inizio alla fine del mio percorso di tesi (testando anche gli esempi da me realizzati), dandomi sempre dei riscontri sinceri e chiari sia sulla parte di teoria e scrittura, che sulla parte pratica di implementazione del framework.

La seconda persona da ringraziare è il professor Alessandro Ricci, nonché relatore di questa tesi, che non solo mi ha aiutato nella stesura della medesima, ma che ho anche avuto modo di avere come professore sia in triennale che magistrale e che è stato in grado di appassionarmi e farmi interessare sempre alle materie da lui trattate. Ancora oggi, penso che il corso di "Sistemi Embedded ed IoT" sia uno dei più belli che abbia seguito, ed è stato proprio questo corso ad introdurmi alla realtà mista e alle sue potenzialità.

Per quanto riguarda il percorso da me affrontato durante gli studi, devo ringraziare la mia famiglia, mia mamma Monica e mio papà Stefano, per avermi sempre sostenuto in tutto quello che ho fatto, per avermi sempre ascoltato in sala da pranzo, quando parlavo di quello che studiavo e che mi piaceva e per essere stati sempre, letteralmente, a una chiamata di distanza quando avevo bisogno di loro; della mia famiglia ringrazio anche mia sorella Sofia, per essere stata un modello di ispirazione nello studio e per essere la mia fan numero uno, qualsiasi cosa faccia o decida di fare.

Infine, devo ringraziare anche altre tre persone che sono state per me importanti durante questo percorso. La prima è Giulia, per avermi spinto a seguire i miei sogni, per aver creduto in me più di quanto io credevo in me stessa, ed essere stata effettivamente quello che Rick dice che devi essere per gli amici. La seconda è Carlotta, perché più di tutti non c'è persona che mi abbia compreso meglio di lei, con cui non ho neanche bisogno di parlare per poter dire quello che penso, sia che sia bello o brutto. La terza è Martina, per avermi ascoltato sempre, aver gioito delle mie vittorie e aver percepito la mia tristezza per le mie sconfitte.

Infine, vorrei chiudere la tesi e i ringraziamenti, con una frase che per me riassume un po' la storia che è sta dietro a questo elaborato e a parte del cammino affrontato:

"The only real failure is the failure to try, and the measure of success is how we cope with disappointment."
Deborah Moggach, The Best Exotic Marigold Hotel

Bibliografia

[1] Babylon. Babylon.js. `https://www.babylonjs.com/`. Accessed June 2023.

[2] Steve Benford, Chris Greenhalgh, Gail Reynard, Chris Brown, and Boriana Koleva. Understanding and constructing shared spaces with mixed-reality boundaries. *ACM Transactions on computer-human interaction (TOCHI)*, 5(3):185–223, 1998.

[3] Irena Cronin and Robert Scoble. *The The Infinite Retina: Spatial Computing, Augmented Reality, and how a collision of new technologies are bringing about the next tech revolution*. Packt Publishing Ltd, 2020.

[4] Croquet. Croquet.io. `https://croquet.io/docs/croquet/`. Accessed June 2023.

[5] Croquet. Model-view-reflector. `https://croquet.io/docs/croquet/tutorial-2_1_model_view_reflector.html`. Accessed June 2023.

[6] Douglas C. Engelbart. Augmenting human intellect: A conceptual framework. *SRI Summary Report AFOSR-3223*, pages 1–6, 1962.

[7] Douglas C Engelbart and William K English. A research center for augmenting human intellect. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 395–410, 1968.

[8] Barrett Ens, Joel Lanir, Anthony Tang, Scott Bateman, Gun Lee, Thammathip Piumsomboon, and Mark Billinghurst. Revisiting collaboration through mixed reality: The evolution of groupware. *International Journal of Human-Computer Studies*, 131:81–98, 2019.

[9] Fabrizio Palmas, Gurdrun Klinker. Defining extended reality training: A long-term definition for all industries. *2020 IEEE 20th International Conference on Advanced Learning Technologies (ICALT)*, pages 323–324, 2020.

[10] Vanessa Fraudenberg. Dls 2020 keynote by vanessa freudenberg: Croquet. a unique collaboration architecture. `https://www.youtube.com/watch?v=ujOVHVAjXj4`. Accessed June 2023.

[11] Epic Games. Unreal engine. `https://www.meta.com/it-it/blog/quest/meta-quest-pro-privacy/`. Accessed August 2023.

[12] Simon Greenwold. Spatial computing. *Massachusetts Institute of Technology, Master*, 2003.

[13] Jonathan Grudin. Computer-supported cooperative work: History and focus. *Computer*, 27(5):19–26, 1994.

[14] Carl Gutwin and Saul Greenberg. A descriptive framework of workspace awareness for real-time groupware. *Computer Supported Cooperative Work (CSCW)*, 11:411–446, 2002.

[15] Wennan He, Mingze Xi, Henry Gardner, Ben Swift, and Matt Adcock. Spatial anchor based indoor asset tracking. In *2021 IEEE Virtual Reality and 3D User Interfaces (VR)*, pages 255–259. IEEE, 2021.

[16] Jon Hindmarsh, Mike Fraser, Christian Heath, Steve Benford, and Chris Greenhalgh. Object-focused interaction in collaborative virtual environments. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 7(4):477–509, 2000.

[17] Hiroshi Ishii, Minoru Kobayashi, and Jonathan Grudin. Integration of interpersonal space and shared workspace: Clearboard design and experiments. *ACM Transactions on Information Systems (TOIS)*, 11(4):349–375, 1993.

[18] Robert Johansen. *GroupWare: Computer Support for Business Teams*. The Free Press, USA, 1988.

[19] Kronos. Webgl. `https://www.khronos.org/webgl/wiki/Main_Page`. Accessed August 2023.

[20] Charlotte P Lee. Boundary negotiating artifacts: Unbinding the routine of boundary objects and embracing chaos in collaborative work. *Computer Supported Cooperative Work (CSCW)*, 16:307–339, 2007.

[21] Charlotte P Lee and Drew Paine. From the matrix to a model of coordinated action (moca) a conceptual framework of and for cscw. In *Proceedings of the 18th ACM conference on computer supported cooperative work & social computing*, pages 179–194, 2015.

[22] TW Malone. What is coordination theory? national science foundation: Coordination theory workshop. *Cambridge, Massachusetts: Massachusetts Institute of Technology*, 1988.

[23] Microsoft. Hololens 2 fundamentals: develop mixed reality applications. `https://learn.microsoft.com/en-us/training/paths/beginner-hololens-2-tutorials/`. Accessed August 2023.

[24] Microsoft. Microsoft mesh overview. `https://learn.microsoft.com/en-us/mesh/overview`. Accessed August 2023.

[25] Microsoft. Shared experiences in mixed reality. `https://learn.microsoft.com/en-us/windows/mixed-reality/design/shared-experiences-in-mixed-reality`. Accessed August 2023.

[26] Microsoft. Unity tytorials. `https://learn.microsoft.com/en-us/windows/mixed-reality/develop/unity/tutorials#hololens-2-tutorials`. Accessed August 2023.

[27] Microsoft. What is an hologram? `https://learn.microsoft.com/en-us/windows/mixed-reality/discover/hologram`. Accessed August 2023.

[28] Paul Milgram, Haruo Takemura, Akira Utsumi, and Fumio Kishino. Augmented reality: a class of displays on the reality-virtuality continuum. *Telemanipulator and Telepresence Technologies*, 2351:282 – 292, 1995.

[29] Mozilla. Foundamentals of webxr. `https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API/Fundamentals`. Accessed June 2023.

[30] Mozilla. The role of frameworks. `https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API/Fundamentals#the_role_of_frameworks`. Accessed June 2023.

[31] Mozilla. Webxr device api. `https://developer.mozilla.org/en-US/docs/Web/API/WebXR_Device_API`. Accessed June 2023.

[32] Kjeld Schmidt and Carla Simonee. Coordination mechanisms: Towards a conceptual foundation of cscw systems design. *Computer Supported Cooperative Work (CSCW)*, 5:155–200, 1996.

[33] Leon D Segal. Effects of checklist interface on non-verbal crew communications. Technical report, 1994.

[34] Shashi Shekhar, Steven K. Feiner, and Walid G. Aref. Spatial computing. *Commun. ACM*, 59(1):72–81, dec 2015.

[35] Investopedia Team. Killer application. `https://www.investopedia.com/terms/k/killerapplication.asp`. Accessed August 2023.

[36] Unity Technologies. Unity. `https://unity.com/`. Accessed August 2023.

[37] Three. Three.js. `https://threejs.org/manual/#en/fundamentals`. Accessed June 2023.

[38] Vocabolario Treccani. Collabolare. `https://www.treccani.it/vocabolario/collaborare/`. Accessed August 2023.

[39] Mirko Viroli. Software quality: overview and techniques (principles, rules, agility, patterns). University Lecture, 2022.

[40] J.M. Zheng, K.W. Chan, and I. Gibson. A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems*, E77-D(12):1321–1329, 1994.

[41] J.M. Zheng, K.W. Chan, and I. Gibson. Virtual reality. *IEEE Potentials*, 17(2):20–23, 1998.