**ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA**
**CAMPUS DI CESENA**

Scuola di Ingegneria e Architettura
Corso di Laurea (Triennale) in Ingegneria e Scienze Informatiche

# Prototipazione di una applicazione collaborativa e distribuita per schermi multitouch

Tesi di laurea in
Programmazione ad oggetti

*Relatore*
**Prof. Mirko Viroli**

*Candidato*
**Federico Bonetti**

*Correlatore*
**Dott. Gianluca Aguzzi**

Quarta sessione di Laurea
Anno Accademico 2022-2023

# Sommario

In questa tesi si descrivono tutti i passaggi per la realizzazione di un prototipo di applicativo multitouch e collaborativo in grado di realizzare grafi. Si comincia con una fase di studio in cui si analizzano le tecnologie usate, si descrivono la libreria scelta per la comunicazione, SignalR e la piattaforma UWP. Si esaminano i possibili tipi di input che può ricevere uno schermo touch, in particolare le gesture, per poi commentare in che modo queste si possano sposare con un software improntato sulla collaborazione tra più utenti.

Da questo studio, si passa poi a descrivere il prodotto finale nei dettagli per quanto riguarda le funzionalità previste (sulla base di queste due tecnologie), come sono state gestite le classi e in che modo è stato implementato il codice.

*Questa è l'unica paginetta che non intendo scrivere in maniera tecnica e fredda, per ringraziare chi mi ha dato una mano a raggiungere l'obbiettivo della laurea. I primi che voglio ringraziare, non possono non essere mia madre, Carla Chesi e mio padre, Massimo Bonetti. Loro mi hanno sostenuto in ogni modo possibile, sia in modo diretto e sostanziale (finanziandomi gli studi e dandomi tutto quello di cui avevo bisogno) che con piccoli gesti, incoraggiandomi nei momenti in cui qualche esame andava male, non pressandomi su questioni di media o tempistiche e, in generale, dimostrandosi sempre presenti qualsiasi problema potessi riscontrare.*

*Mi ricordo quando non passai il primo esame, mia madre mi disse: "Federico, se molli per così poco non ti riconosco, sono tranquilla che ce la farai". Un altro momento è quando si verificò un problema tecnico con la mia carta di credito a pochi giorni di distanza dall'ultimo esame che avrei sostenuto e, mio padre, di sua iniziativa si fece qualche oretta di macchina per rifornirmi di denaro.*

*Proseguo ringraziando il professor Mirko Viroli per avermi dato l'opportunità di realizzare questa tesi e il tutor che mi ha seguito durante lo svolgimento, il dottor Gianluca Aguzzi, per l'importante aiuto. Il tutor non si è limitato al suo mero compito di supervisore, ma ha risposto a tutti i miei dubbi in ambito informatico, arricchendo le mie conoscenze a dovere nel corso di questa esperienza.*

*Dulcis in fundo, ringrazio tutti i miei amici per i piccoli e grandi gesti, perchè anche staccare la spina dagli studi i weekend è fondamentale per ripartire con più grinta. Non in ordine di importanza, ringrazio i miei amici di lunga data Marco Magagnoli, Joy Franciosi, Vincenzo Savino e Raffaela Cetronio, ringrazio mia sorella Giorgia Bonetti e le sua miche, i miei amici più recenti conosciuti a Cesena Andrei Gabriel Iulian Mohanu, Luca Cantagallo e Salvatore Zammataro e i miei coinqulini Tomasso La Selva, Edoardo Manzini e Yasir Touijer.*

# Indice

**Sommario** **iii**

**Introduzione** **xiii**

**1 Background** **1**
1.1 Multitouch e Gesture . . . . . 1
1.2 Software collaborativi . . . . . 3
1.2.1 Multiutente . . . . . 5
1.2.2 Spazio d'interazione comune . . . . . 6
1.2.3 Interazione in tempo reale . . . . . 7
1.2.4 Applicazioni collaborative conosciute . . . . . 8
1.3 Universal Windows Platform . . . . . 9
1.4 SignalR . . . . . 11

**2 Requisiti** **15**
2.1 Requisiti funzionali . . . . . 15
2.2 Requisiti non funzionali . . . . . 17

**3 Design** **19**
3.1 UML . . . . . 20

**4 Implementazione e validazione** **23**
4.1 Implementazione del concetto di nodo . . . . . 23
4.2 Pagina principale . . . . . 25
4.3 Implementazione del concetto di Arco . . . . . 29
4.4 Gesture di manipolazione . . . . . 31
4.5 Comunicazione tra LIM e smart desk . . . . . 34

**5 Conclusioni** **39**

# Elenco delle figure


1.1 Gesture principali usate in schermi multitouch . . . . . 2
1.2 API di UWP . . . . . 10
1.3 immagine esplicativa di RPC dalla documentazione ufficiale microsoft 12
1.4 Ciclo di vita di una istanza della classe Hub . . . . . 14

3.1 Il diagramma UML completo . . . . . 21
3.2 Il diagramma UML relativo alle classi per la creazione del grafo . . 21
3.3 Il diagramma UML relativo alla gestione del touch e delle gesture . 22
3.4 Il diagramma UML relativo alla gestione della connessione . . . . . 22

4.1 un nodo disegnato dal codice sovrastante . . . . . 24
4.2 un nodo selezionato, colorato dal codice sovrastante . . . . . 24
4.3 schermata principale senza aver inserito nessun elemento . . . . . 25
4.4 Due nodi collegati da un arco . . . . . 30
4.5 due nodi, quello a destra è stato soggetto ad una trasformazione di scalatura dovuta alla gesture di stretch . . . . . 32
4.6 due nodi, quello a destra è stato soggetto ad una trasformazione di rotazione dovuta alla gesture di turn . . . . . 32
4.7 esempio di grafo, disegnabile tramite questa applicazione . . . . . 35
4.8 la classe Inertia e la comunicazione traimte SignalR, permettono di lancira un nodo da uno schermo all'altro . . . . . 38

# Elenco dei listati


4.1 Disegnare il nodo su UWP . . . . . 23
4.2 Cambio di colori al nodo per renderlo selezionato . . . . . 24
4.3 Finestra principale del progetto, in XAML . . . . . 26
4.4 Creazione e aggiunta di un nodo alla griglia . . . . . 26
4.5 Funzione “OnTapped”, scatenata dall’evento “PointerPressed” . . . 27
4.6 Funzione “OnTapped”, scatenata dall’evento “PointerPressed” . . . 28
4.7 Creazione grafica di un arco, dal costruttore di “ArchView” . . . . . 29
4.8 Rimozione di un nodo e, quindi, dei relativi archi collegati . . . . . 30
4.9 Impostare l’origine della trasformazione di scalatura al centro del nodo . . . . . 33
4.10 Risposta alla gesture di swipe, tramite la trasformazione di traslazione 33
4.11 Sistema di mapping tra la View e il Model, a seconda del dispositivo 34
4.12 Ricezioni messaggi . . . . . 36

# Introduzione

Questa tesi è dedicata allo studio delle tecnologie multitouch e dei software collaborativi. L'obbiettivo finale è la realizzazione di un applicativo che permetta all'utente di creare grafi da due dispositivi con schermo tattile e che dia la possibilità agli utenti di lavorare in contemporanea sulla stessa area di lavoro virtuale.

Si rende necessario descrivere bene queste tecnologie, chiarire che cosa intendiamo per schermo multitouch e quali tipi di input possiamo affibbiargli (tocchi in più punti, gesture), illustrare le differenze tra un software che si limita a comunicare in tempo reale e un applicativo collaborativo (che quindi rende disponibile un'area di lavoro virtuale accessibile da più utenti). Un'applicazione che permette esclusivamente di chattare, ad esempio, non è definibile come software collaborativo.

I software collaborativi stanno sempre diventando più comuni, visto la praticità di lavorare in contemporanea con qualcuno fisicamente distante da te su uno stesso file o elemento. Un grande merito della diffusione di questi programmi va ai progressi che continuiamo a compiere in ambito di infrastrutture di reti informatiche. Software di questo tipo infatti scambiano continuamente messaggi, gestire una mole di pacchetti così importante qualche decennio fa sarebbe stato molto più problematico.

Per cercare di esplicitare meglio l'importanza di questa tecnologia e lasciare intendere quanto in realtà siano già in uso i software collaborativi, in questa tesi una sessione è dedicata esclusivamente ad esempi di applicazioni comuni che vengono usate quotidianamente da molti utenti. Capire quanto sia frequente usare questi programmi, ci permette anche di fare riflessioni su quali si svilupperanno in un futuro prossimo e su quanto il concetto di software collaborativo possa confluire bene con l'idea di "Internet of Things".

Per la realizzazione del prototipo, si è scelto di adoperare la piattaforma UWP (Universal Windows Platform), che permette al programmatore di creare il software direttamente per tutti i dispositivi che supportano un sistema operativo Windows moderno. Questo significa che alcune funzionalità del programma potranno essere testate anche da un comune pc).

Per quanto riguarda la comunicazione si userà la libreria SignalR, che parte

col presupposto di inviare ogni messaggio a tutti i destinatari e di sfruttare la tecnologia delle Remote Procedure Call (abbreviabile in RPC), ovvero chiamare le funzioni descritte sul server direttamente dal client.

Lo studio di UWP e di SignalR sarà di aiuto per capire quali funzioni multitouch e collaborative potranno essere sviluppate nel limite di tempo imposto e quali calzano bene per la realizzazione e modifica di un grafo.

Gli eventi delle gesture multitouch che mette a disposizione UWP sono facili da associare alle trasformazioni geometriche. Questo perchè, con il diffondersi degli smartphone, è diventata ormai consuetudine compiere alcuni gesti coordinati con le dita per "allargare", "restringere" o "ruotare" una immagine sul proprio dispositivo.

A partire dallo studio di queste, si possono pensare a diverse funzionalità da implementare per quanto riguarda il lato interazione utente/nodo per mezzo del touchscreen.

La tesi verrà strutturata come segue:

- Background: si descrivono la tecnologia multitouch e i software collaborativi, completando con esempi e possibili sviluppi futuri di queste.
- Requisiti: si decidono i requisiti che il prototipo che si andrà a realizzare dovrà soddisfare, distinguendoli tra funzionali e non funzionali.
- Design: si progetta il prototipo, scendendo a livello di struttura delle classi, con l'ausilio di diagrammi UML.
- Implementazione e validazione: si espone la soluzione adottata nel dettaglio, citando le righe di codice adottate e spiegandone il funzionamento, nonché mostrando il risultato per merito di immagini del software in esecuzione.

# Capitolo 1

# Background

## 1.1 Multitouch e Gesture

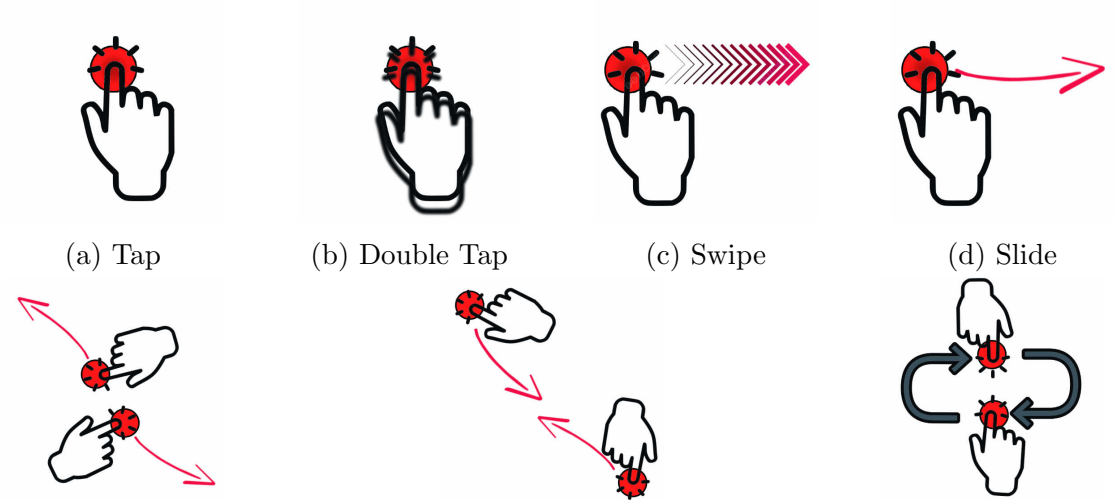
L'avanzamento tecnologico ha reso sempre più comune l'utilizzo di schermi tattili (touchscreen) e, di conseguenza, delle relative funzionalità con le quali siamo abituati ad approcciarci a questi dispositivi. I primi calcolatori con schermo touchscreen funzionante risalgono alla fine degli anni '60, questi erano in grado esclusivamente di ricevere in input un'informazione data da un singolo tocco. Una naturale evoluzione di questo tipo di dispositivi, è stata permettere agli schermi associati di poter rilevare più pressioni contemporaneamente in punti distinti; tra gli anni '80/'90 nascono i primi schermi multitouch.

Le potenzialità di poter usare schermi multitouch hanno fatto si che si cominciassero a sviluppare calcolatori che non prevedano l'utilizzo di pulsanti o di altri elementi hardware per dare input, tanto che oggigiorno tutti i dispositivi cellulari o tablet (moderni) dispongono di questa tecnologia. Le tastiere dei computer sugli attuali dispositivi touchscreen possono essere sostituiti da pulsanti virtuali che vengono generati sullo schermo e il click di un comune "mouse" può essere sostituito da un tocco.

Il multitouch permette d'implementare delle "gesture", ovvero movimenti coordinati delle pressioni sullo schermo, per dare input specifici; è infatti oramai diventata consuetudine per tutti ruotare un'immagine sul proprio smartphone muovendo la pressione di almeno 2 dita sullo schermo in senso orario o antiorario. Questo input è una gesture.

Le gesture quindi non sono spostamenti casuali delle dita, ma sono movimenti coordinati che possono essere identificati in modo distinto l'uno dall'altro. Quelle più comuni che possiamo riconoscere, elencate anche sulla documentazione ufficiale Microsoft relative alla tecnologia UWP (Universal Windows Platform) [1], sono

[1] documentazione ufficiale Microsoft UWP [1]



(a) Tap (b) Double Tap (c) Swipe (d) Slide

(e) Stretch (f) Pinch (g) Turn

Figura 1.1: Gesture principali usate in schermi multitouch

(Figura 1.1):

- Tap (Figura 1.1a): singolo tocco rapido.
- DoubleTap (Figura 1.1b): 2 tocchi veloci.
- Hold: tocco prolungato.
- Swipe (Figura 1.1c): movimento veloce di una pressione che parte da un punto e termina in un altro.
- Slide (Figura 1.1d): movimento prolungato di una pressione che parte da un punto e termina in un altro.
- Turn (Figura 1.1g): movimento ad almeno 2 dita, che, scelto un punto nello schermo, queste ci ruotano intorno in senso orario/antiorario.
- Pinch (Figura 1.1f): 2 dita messe in punti speculari, tra loro si avvicinano.

- Stretch (Figura 1.1e): 2 dita messe in punti speculari, tra loro si allontanano.

Essendo le gesture movimenti che compie l'uomo, con la mano solitamente, bisogna tener conto che le pressioni applicate a schermo non si coordineranno in maniera perfetta, compiendo gesti che seguano uno schema matematico in maniera esatta. Per riconoscerle e distinguerle l'una dall'altra, si analizzano quindi i seguenti fattori:

- Tempo di esecuzione
- Accuratezza (è impossibile che sia un movimento perfetto, assente di imprecisioni)
- Occlusione
- Target sul quale si applica la gesture

## 1.2 Software collaborativi

Un altro protagonista di questa tesi è lo studio di reti di software collaborativi, ovvero **client diversi** che agiscono su **un'unica area di lavoro virtuale**, in contemporanea.

Dagli anni '60, la tecnologia che permette a più computer di far parte di una rete e di condividere risorse, continua a evolversi, usando sempre strategie più studiate. Ci si vantava dei primi successi dell'intelligenza artificiale, come viene riportato dalla rivista "Archeologia informatica" riguardo al supercomputer "Maniac 1" del 1953; [2] "L'architettura è talmente moderna da renderlo persino famigliare (sebbene primitivo nell'interazione e nella tecnologia). ... Quando non svolgeva calcoli per le bombe atomiche, Maniac I trovò il tempo -anche- di divertirsi e, nel 1956, fu il primo computer in grado di battere un uomo a scacchi con una variante del "Los Alamos Chess" (una scacchiera 6×6 priva degli alfieri)".

La prima vera e propria rete di computer venne progettata dall'esercito americano, come racconta Giovanni Natalini nel suo articolo [3]: "Nel 1958, nel pieno della guerra fredda, gli USA fondarono l'istituto di ricerca ARPA (poi DARPA). La sua missione era quella di trovare soluzioni tecnologiche innovative a un certo numero di problemi (principalmente militari)". Questo progetto trovò un vero risultato concreto solo alla fine degli anni '60, nel progetto americano ARPANET, la prima traccia di Internet nella storia.

Le prime reti concettualmente erano molto semplici: prevedevano un calcolatore centrale, nominato "mainframe", che, tramite periferiche fisiche, si connetteva con terminali multipli, composti solo da tastiera e monitor.

Grazie al fatto che le periferiche fisiche di collegamento sono state sostituite da segnali digitali e analogici e i computer hanno sempre più raddoppiato la loro potenza di calcolo di anno in anno, intorno agli anni '90 sono entrate sempre più in voga le reti distribuite. Questo tipo di rete, al contrario delle precedenti centralizzate, prevede che ogni terminale sia un'entità autonoma, capace di elaborare dati e compiere calcoli. I terminali tra di loro si scambiano messaggi, o per meglio dire pacchetti (gruppi di dati), che vengono prima digitalizzati e poi spediti tramite gli appositi protocolli.

I software collaborativi a scambio di messaggi in tempo reale, sono possibili grazie ai sistemi distribuiti, che permettono di avere latenza minima tra un messaggio e l'altro. Quando si sviluppano software di questo tipo, il primo passo è avere chiaro quale tipo di architettura di rete si vuole creare. A livello 3 della rete, le principali possibili architetture sono:

- Client-Server: i client condividono risorse sul server. I primi, a livello di rete sono solo in grado d'inviare richieste a quest'ultimo, che prende le vere e proprie decisione e gestisce le risorse.

- Peer to peer: in questo caso, ogni dispositivo è in grado d'inviare e ricevere dati dagli altri. La particolarità di questa architettura è che non esiste una gerarchia tra i client.

Un software collaborativo non si identifica solo dal fatto di avere un'area di lavoro comune tra client diversi, ma anche dalla particolarità che questi **debbano scambiarsi messaggi in tempo reale**, aggiornarsi e tenersi pronti per inviare o riceve informazioni in qualsiasi momento.

Una delle funzionalità più importanti da gestire di questi software, è trovare una strategia vincente per tenere tutti i client sincronizzati. Non deve succedere infatti che, ad esempio, se l'area di lavoro virtuale presenta degli elementi, un client non rimanga aggiornato e sia ignaro dell'esistenza di alcuni di questi, o viceversa, che ne crei di più rispetto a quanto dovrebbe, dovuto al fatto di messaggi multipli gestiti in modo errato.

Uno dei fattori che decreta la "strategia vincente", è dovuta da come si scambiano i messaggi i client, detta in maniera più tecnica, dal protocollo di rete. A livello 4 della rete i protocolli più usati sono:

- TCP: il mittente e il destinatario prima di comunicare effettuano l'"handshake", ovvero uno scambio di messaggi iniziale per garantire che la connessione venga instaurata correttamente. Il protocollo controlla che i messaggi arrivino a destinazione interi e una sola volta, senza errori, grazie al suo sistema di controllo sui bit frazionati consegnati.

- UDP: questo protocollo invece non effettua l'"handshake" e non controlla il corretto ricevimento del messaggio da parte del destinatario, in compenso è più veloce. Viene quindi utilizzato prevalentemente per applicazioni dove conta ricevere una grande mole di messaggi tempestivamente e non che ogni messaggio sia corretto.

In questo capitolo abbiamo quindi individuato i tre principali fattori di un'applicazione collaborativa che sono: *multi utenza*, comunicazione *real-time*, e area di lavoro virtuale *comune*.

### 1.2.1 Multiutente

Per software multiutente intendiamo la possibilità di un programma di far accedere a risorse comuni più utenti. Software di questo tipo possono permettere agli utenti d'interagire tra loro in tempo reale, o di lavorare su uno stesso sistema in momenti diversi (e.g., siti web che permettono la registrazione). In un software collaborativo, genericamente si tratta di utenti che lavorano da computer diversi tra loro e che quindi possiamo anche identificare come client.

Gli utenti non è detto che abbiano tutti lo stesso livello di privilegi tra loro, certe applicazioni necessitano una distinzione tra amministratori e utente semplice. Se, ad esempio, parliamo di un software collaborativo che simula un'aula di scuola virtuale, è bene che il professore venga identificato come amministratore e che abbia più privilegi rispetto agli alunni (utenti semplici). Potrebbe essere utile per l'insegnante avere la possibilità di silenziare gli studenti, di poter invitare altri utenti esterni durante la lezione, o di condividere il proprio schermo con gli alunni. Si potrebbe prevedere invece che gli studenti debbano chiedere il consenso al prof anche a livello informatico, per compiere azioni di questo tipo.

Uno dei sistemi multiutente più famoso è Unix. Questo sistema operativo, nonostante sia nato nel 1969, secondo la rivista "Managed Server", continuerà a essere utilizzato a lungo [4]: "Unix e le sue varianti continuano a funzionare su un'ampia gamma di sistemi, incluse workstation, server e supercomputer. Linux, in particolare, ha preso il comando nelle implementazioni simili a Unix, guadagnando una forte presenza nei data center e sulle piattaforme cloud. Inoltre, il sistema operativo ora funziona su tutti i 500 migliori supercomputer del mondo".

Sistemi basati su Unix permettono di attribuire ai file permessi per ogni utente, identificando cosa quell'utente può e non può fare su quel file. Nella fattispecie, i permessi di Unix sono:

- Lettura: permettere di leggere il contenuto del file.
- Scrittura: permettere di leggere e modificare il contenuto del file.

- Esecuzione: permettere di eseguire il file se è un eseguibile.

In Unix, è possibile associare i permessi a un gruppo di utenti nominato classe, invece che singolarmente. Per ogni file, gli utenti vengono suddivisi nel seguente modo:

- Proprietario: solitamente il creatore del file.
- Gruppo di appartenenza: a questo, appartengono tutti gli utenti dello stesso gruppo del proprietario del file.
- Tutti gli altri utenti.

### 1.2.2 Spazio d'interazione comune

Definito che abbiamo un software multiutente che lavora su una certa infrastruttura di rete, bisogna decretare un'area virtuale di memoria dove registrare e leggere i dati d'interesse comune. L'accesso a un'area di memoria condivisa tra più utenti, o, per meglio dire, tra più processi, viene definito come sistema concorrente, per la caratteristica che questi possono tentare di accedere a una stessa risorsa condivisa nello stesso momento. Gli errori derivati da sistemi concorrenti mal gestiti sono noti:

- Corse critiche: data una stessa risorsa condivisa, si verifica se due o più processi cercano di accederci contemporaneamente. Il risultato è imprevedibile, il processo successivo potrebbe non aver fatto in tempo a vedere la modifica del primo o potrebbero effettuare la modifica nello stesso momento, potendo anche aggiornare la risorsa in modo corrotto.
- Stallo (deadlock): può verificarsi quando, per mal gestione, ogni processo attende un evento che si può verificare soltanto grazie a un altro processo. In questo modo ogni processo attende all'infinito che gli altri sblocchino la risorsa, ma non avverrà mai essendo tutti bloccati.
- Starvation: quando i processi hanno livelli di priorità diversi per l'accesso a una risorsa, può essere che quello con priorità minore rimanga bloccato anche se non si verifica una situazione di stallo.

Essendo noti i problemi derivati da sistemi concorrenti, si sono studiate tattiche per prevenire questi comportamenti indesiderati:

- Mutua esclusione (mutex): operazioni atomiche di blocco o sblocco sulle risorse condivise, per rendere temporaneamente l'accesso esclusivo a un processo.

- Scambio di messaggi: i processi si scambiano messaggi senza scomodare le risorse in comune.

- Condition variables: gestione tramite due funzioni generalmente definite come “wait” e “signal”. La prima mette un processo in attesa indefinita, la seconda lo risveglia.

- Semaforo: variabile globale e atomica, incrementabile e decrementabile, che sblocca o blocca i processi a seconda del suo valore.

### 1.2.3 Interazione in tempo reale

Per comunicazione in tempo reale intendiamo una qualsiasi forma di comunicazione che permette di scambiarsi messaggi a bassa latenza. Gli esempi più lampanti di comunicazione in tempo reale sono i software di videochiamata o di chat, che si distinguono dai programmi di posta elettronica o dai commenti di foto o video sui social.

Nei programmi di comunicazione in tempo reale, la priorità viene data dalla tempestività nel ricevere un'informazione dai client ed elaborarla, più che dalla correttezza di quest'ultima al millesimo per ogni scambio di messaggi. Prendendo il caso delle videochiamate, l'utente preferisce continuare a poter parlare con il proprio interlocutore a costo di perdere qualche frame ogni tanto se la connessione non è perfetta, piuttosto che aspettare che tutte le singole immagini del video vengano inviate perfette a risoluzione elevata e quindi rendere la latenza potenzialmente (o per meglio dire, probabilmente) rilevante.

La mancanza di precisione delle informazioni passate per ogni singolo messaggio, rimane comunque un difetto che il programmatore deve prevedere e gestire, per non permettere una mancanza di sincronizzazione rilevante tra i client.

Prendiamo come esempio un personaggio di un videogioco, che ogni volta che si muove deve passare informazioni rilevanti riguardante i suoi spostamenti. Se si decide di gestire la situazione passando come valore l'incremento di posizione per ogni singolo istante, si rischia di incorrere in problemi di incoerenza delle informazioni tra i client, ad esempio il personaggio potrebbe essersi spostato 5 volte di un valore 6, quindi in totale di 30. Se si perde un messaggio per strada il personaggio, secondo gli altri client, si sarebbe spostato invece di 24... a lungo andare un sistema di questo tipo può generare errori grossolani. Se invece si decide di passare la posizione assoluta, anche se viene perso un messaggio ogni tanto, il successivo correggerà il sistema aggiornando correttamente il punto in cui si trova il personaggio e quindi risincronizzando i client.

### 1.2.4 Applicazioni collaborative conosciute

Una delle applicazioni collaborative più comune sono i fogli di Google, ovvero fogli di calcolo a cui possono accedere più utenti in contemporanea da dispositivi diversi.

Per dire quanto sono in voga queste app, anche il software su cui è stata scritta questa tesi "Overleaf" [2] è collaborativo. Permette infatti di scrivere un testo e impaginarlo da codice, con il linguaggio "LaTeX". In questa app, più utenti possono scrivere in contemporanea e commentare righe di codice tramite dei "post-it" per lasciare appunti da far leggere ai propri colleghi.

Passando a settori specifici, l'utilizzo delle applicazioni collaborative in tempo reale può avere molteplici scopi. Ad esempio un ristorante può servirsi di un suo applicativo per gli ordini, che permetta ad esempio ai camerieri di informare i cuochi su quali piatti vadano preparati.

Un'applicazione collaborativa nota ai team di sviluppo, è Trello, che si pone con l'obbiettivo di semplificare la gestione della divisione dei compiti a questi ultimi. Gli utenti creano le schede di lavoro e se le spartiscono tra di loro. Un utente che viene associato a una scheda deve indicare se quel lavoro è in atto o concluso.

Sui software che nascono per creare mappe concettuali sta sempre più prendendo piede il concetto di "collaborativo", visto che è utile condividerli tra compagni di studio e modificarli in contemporanea anche a distanza. Un'App nata con questo scopo è "Miro"[3], che permette a più utenti connessi su uno stesso foglio di lavoro di creare schemi, mappe concettuali o disegni.

Il concetto di applicazione IoT solitamente si sposa bene con il concetto di applicazione collaborativa. Una casa domotica può essere gestita da tutti coloro che hanno l'app apposita e il permesso di interagire con l'abitazione, oppure, pensando più in grande, una fabbrica che si rispecchia nei parametri dell'industria 4.0 probabilmente sarà gestita da più utenti.

Si parla sempre più di wearable computer (computer indossabile), ovvero strumenti come braccialetti, cappelli o comunque calzature, che contengono dei chip, capaci di inviare dati (come ad esempio battito cardiaco, spostamenti di chi li indossa). I wearable computer possono sembrare quasi fantascientifici ai meno addetti ai lavori, la verità è che si tratta di una tecnologia che si studia da decenni. Ad esempio già nel 2006 alla conferenza "International Symposium on Wearable Computers", si parlava dell'importanza di usare questi sistemi nel modo appropriato: [5]"Un design inappropriato può significare che chi lo indossa non è in grado di eseguire compiti specifici o raggiungere obiettivi. Uno stress eccessivo sul corpo può provocare percezioni di disagio, che a loro volta possono influire sulle prestazioni del lavoro, e alla fine sollevano problemi di salute e sicurezza". Nell'ambito della medicina, un paziente in ospedale potrebbe trasmettere dati per merito di

[2]sito di Overleaf
[3]sito di Miro

un wearable computer, senza che lui debba fare nulla e l'equipe di medici che lo segue riceverli, rimanendo aggiornati in qualunque momento sulla sua salute.

Infine, banalmente, tutti i videogiochi multigiocatore online sono applicazioni collaborative. Tutti i giochi sparatutto in multiplayer devono tener conto degli spostamenti degli avatar di tutti i giocatori o insomma delle azioni che stanno compiendo.

## 1.3 Universal Windows Platform

Vista la volontà di Microsoft di voler espandere i dispositivi su cui è possibile usare il sistema operativo Windows (esempio Windows Phone, dispositivi all in one), questa ha deciso di mettere a disposizione dei propri programmatori una piattaforma per poter creare app che girino direttamente su tutti i propri calcolatori.

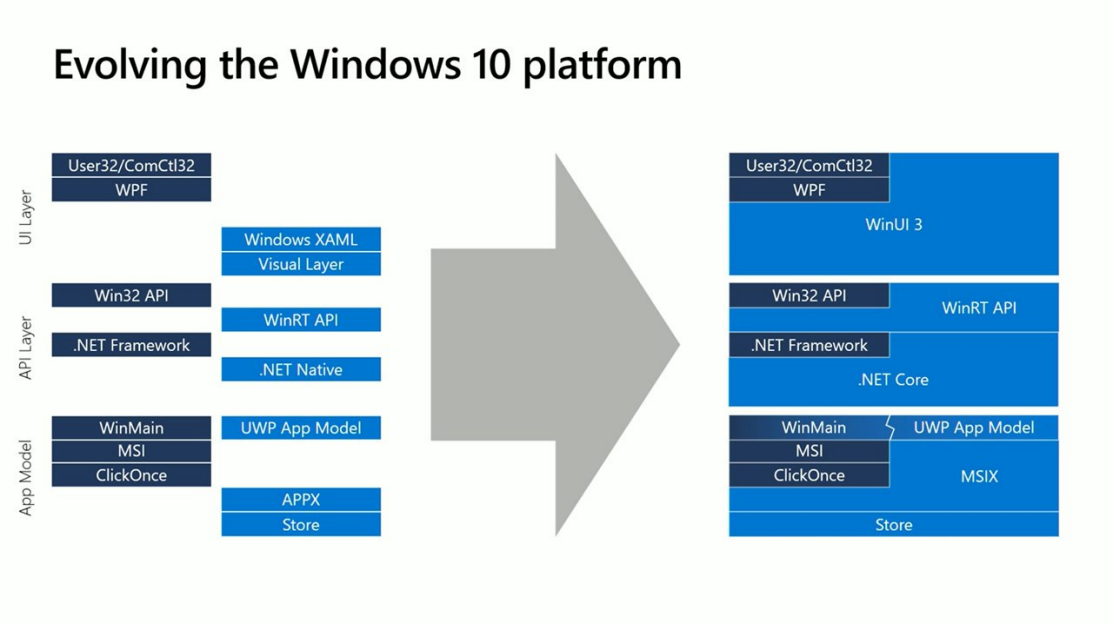
Questa piattaforma si chiama Universal Windows Platform (abbreviabile in UWP), e supporta i linguaggi di programmazione C#, C++, Visual Basic e JavaScript. La documentazione ufficiale di Microsoft, descrive questa piattaforma [6]"come uno dei numerosi modi per creare applicazioni client per Windows. Le app UWP usano le API WinRT per offrire potenti funzionalità avanzate di interfaccia utente e asincrone, ideali per i dispositivi connessi a Internet".

Questa tecnologia in un primo momento ha riscontrato molte critiche da parte dei programmatori che si sono affievolite in seguito a un aggiornamento della piattaforma. Su windowsBlogItalia, Alexandre Milli prova a spiegare il motivo, dal suo punto di vista [7]"Microsoft sta integrando e unificando alcune componenti per facilitare lo sviluppo delle app, come l'interfaccia, le API e il tipo di pacchetto dell'app".

Per la sua caratteristica di poter essere eseguita su qualsiasi tipo di dispositivo che ha un sistema operativo Windows moderno, permette d'implementare metodi per l'utilizzo del touchscreen e si preoccupa che l'interfaccia utente si adatti allo schermo sul quale viene eseguita l'applicazione. Gli elementi quali pulsanti o dispositivi di scorrimento, si adattano al tipo di densità (DPI) e alle dimensioni dello schermo o, comunque, il programmatore può avvalersi di una progettazione reattiva per i layout.

Anche il tipo di input viene gestito prendendo in considerazione che potrebbe variare a seconda del dispositivo. In particolare, un evento "OnPressed" di questa piattaforma può essere scatenato da un "tap" se ci troviamo su uno schermo touchscreen, oppure da un "click" se il dispositivo ha a disposizione un mouse, proprio per permettere all'applicazione di adattarsi a un qualsiasi di questi dispositivi con il minimo sforzo da parte del programmatore.

Figura 1.2: API di UWP




UWP permette anche l'implementazione delle gesture, mettendo a disposizione le sue app anche per dispositivi touchscreen. Non in tutti i casi però queste sono traducibili in gesti applicabili dal mouse.

Nel dettaglio:

- Tap → Singolo click.
- DoubleTap → Doppio click.
- Hold → Click prolungato.
- Swipe → Movimento veloce tenendo premuta il tasto sinistro del mouse che comincia in un punto e si rilascia in un altro.
- Slide → Movimento prolungato tenendo premuta il tasto sinistro del mouse che comincia in un punto e si rilascia in un altro.
- Rotation → Non traducibile
- Pinch → Non traducibile
- Stretch → Non traducibile

Per applicare le gesture, UWP predispone una serie di eventi che si possono associare o sui singoli elementi o sull'intero schermo, che vengono scatenati dalla gesture corrispondente.

Si dividono in tre tipologie: quelli che si attivano appena viene rilevata la pressione e quindi la gesture comincia, quelli che si attivano mentre la gesture è in atto e quelli che si attivano quando viene tolta la pressione e la gesture quindi si conclude.

Gli eventi per rilevare le gesture che UWP mette a disposizione sono:

- PointerPressed: si scatena appena viene rilevato un tocco.
- PointerReleased: si scatena appena viene rilevata la fine di una pressione.
- ManipulationStarted: si scatena quando una gesture che comincia come gesto per effettuare una traslazione, scalatura o rotazione viene rilevata.
- ManipulationDelta: si scatena quando la gesture è in atto. Uno degli argomenti che riceve in input del tipo "ManipulationDeltaRoutedEventArgs" restituisce informazioni relative a se si sta effettuando una delle tipiche gesture di traslazione, scalatura o rotazione e con quale delta.

Altri strumenti utili messi a disposizione da UWP riguardo alle gesture, sono la classe Transform e le relative figlie.

Questa classe può applicare una trasformazione geometrica (in particolare di scalatura, rotazione o traslazione) a un elemento; risultano molto comode se applicate in risposta all'evento ManipulationDelta, in quanto quest'ultimo, come esplicitato in precedenza, riceve come argomento in input "ManipulationDeltaRoutedEventArgs", che può darci un valore delta che indica quanto si siano spostate le pressioni nel compiere la relativa gesture e in che direzione.

Degna di nota anche la funzionalità "Inertia"; questa su un elemento su cui applichiamo una trasformazione, in risposta a un evento scatenato da una gesture, simula la fisica.

Ad esempio se trasliamo un elemento velocemente verso una direzione, una volta tolta la pressione delle dita, questo continuerà a muoversi in quel verso simulando di aver ricevuto una "spinta". Lo stesso concetto si applica ovviamente anche alla rotazione e alla scalatura.

## 1.4 SignalR

Nella documentazione ufficiale di Microsoft, SignalR si presenta così[8]: "è una libreria open source SignalR che semplifica l'aggiunta di funzionalità Web in tempo

reale alle app. La funzionalità Web in tempo reale consente al codice lato server di eseguire il push istantaneo del contenuto nei client".

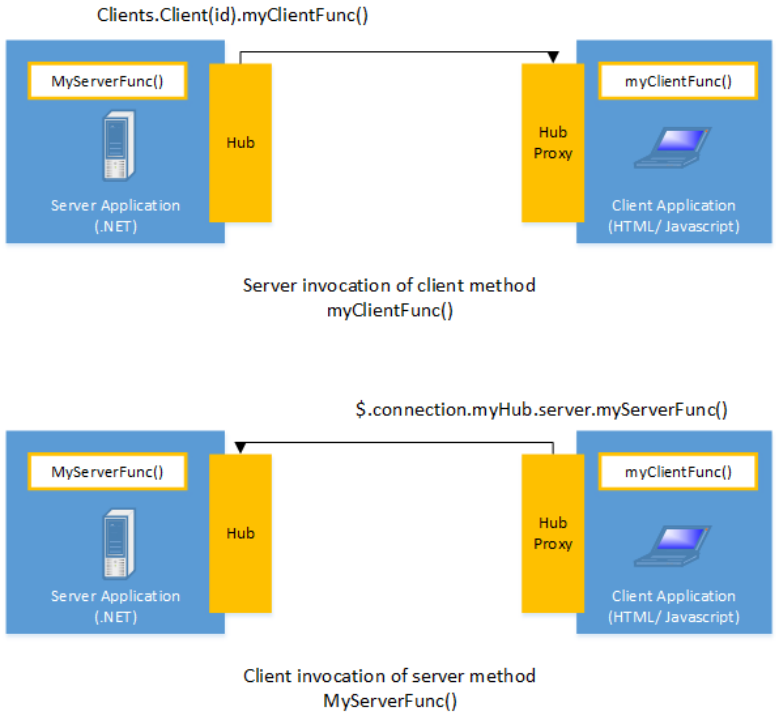
SignalR è una libreria per ASP.NET che permette d'implementare funzionalità Web in tempo reale, programmata per essere usata sui linguaggi C++, C# e JavaScript.

Usufruisce della tecnologia di trasporto delle nuove Web Socket, che forniscono canali di comunicazione full-duplex (connessione tra due client, permessa in entrambe le direzioni, simultaneamente), con protocolli che si basano su TCP.

La particolarità di SignalR è che parte con l'intenzione di facilitare la trasmissione di dati a tutti i client connessi (tramite broadcast).

Nello specifico quindi, il server esegue immediatamente il push del contenuto del codice ai client connessi non appena diventano disponibili, potendo usufruire delle RPC, ovvero chiamare le funzioni direttamente sui client.

Figura 1.3: immagine esplicativa di RPC dalla documentazione ufficiale microsoft




Possiamo identificare 2 modelli per le comunicazioni in SignalR, connessioni persistenti e Hub:

- Connessioni persistenti: endpoint semplice, per la trasmissione a destinatari singoli o raggruppati. Permette di lavorare sui protocolli di comunicazione a basso livello.

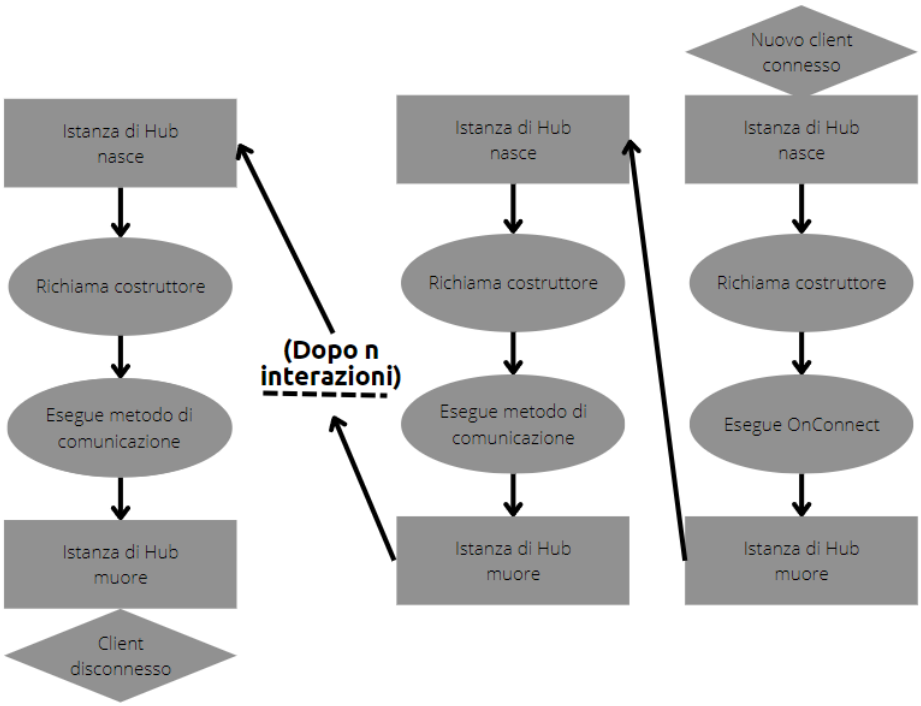
- Hub: pipeline generale di connessione che permette al server di chiamare direttamente i metodi sui client, per merito delle RPC. Si tratta quindi di una programmazione più ad alto livello.

Chiarito che SignalR nasce con l'intenzione di mandare il messaggio a tutti i client connessi, paradossalmente inviare un dato a un singolo terminale diventa più ostico. Per mandare messaggi a client specifici si possono aggiungere dei "filtri"; il metodo pensato dai creatori della libreria è di dividerli in gruppi gestiti dalla classe "Group".

L'API di SignalR definisce i metodi sul server che possono essere chiamati dai client.

Volendo entrare nello specifico della classe Hub in SignalR per linguaggio C#, possiamo analizzare che una sua istanza ha vita breve: viene creata, si mette in attesa di un messaggio, quando lo riceve reagisce eseguendo le istruzioni che le sono state date, poi muore. Per questo motivo, per ogni messaggio inviato, vengono rieseguite anche le istruzioni presenti nel costruttore. Le istruzioni che vengono eseguite alla prima connessione del client (ad esempio quelle che possiamo programmare nel metodo "OnConnected") appartengono a un ciclo differente di vita rispetto all'invio del primo messaggio. Funzionando così, è quindi chiaro che all'interno della classe figlia di "hub" non ci sarà possibile registrare informazioni che abbiano vita più lunga di uno scambio di messaggi. Per ovviare a questo problema, possiamo usufruire del pattern "Singleton", creando un'istanza di una classe che memorizzi i dati che vengono richiamati da "Hub".

Figura 1.4: Ciclo di vita di una istanza della classe Hub



# Capitolo 2

# Requisiti

L'obbiettivo della tesi è quello di sviluppare un'applicazione collaborativa e distribuita per schermi multitouch, che permetta agli utenti di creare dei grafi.

Per grafi in matematica si intende una struttura composta da un insieme di punti nominati "nodi" e un insieme di linee nominate "archi" che uniscono coppie di nodi tra loro.

Nel laboratorio "E" del campus di Cesena (sede distaccata UNIBO) mi sono stati messi a disposizione per questo progetto uno smart desk e una lavagna LIM. Lo smart desk è un tavolo interattivo, avente schermo touchscreen e un hardware sufficiente per supportare un sistema operativo Windows 10. Si può usare come se fosse un classico computer. La LIM ("Lavagna Interattiva Multimediale") ha pressoché le stesse caratteristiche e funzioni dello smart desk, però lo schermo è disposto verticalmente attaccato al muro, ricordando per forma una lavagna.

L'idea è quella di permettere ai due dispositivi di condividere lo stesso "piano di lavoro" diventando di fatto uno l'estensione dello schermo dell'altro, permettendo all'utente di collegare con degli archi i nodi tra i due dispositivi o semplicemente di spostarli, passandoli tra uno schermo e l'altro.

Gli schermi della LIM e dello smart desk sono touchscreen, è possibile sfruttare questa caratteristica per predisporre delle gesture come input per comandi specifici.

## 2.1 Requisiti funzionali

f1: Possibilità per l'utente di scegliere in che tipo di modalità lavorare se in "Inserimento nodi/modifica", se in "Cancellazione" o in "Selezione/inserimento archi".

f2: Possibilità di disegnare un nodo in seguito alla pressione di un dito sul touchscreen, se il software è in modalità "Inserimento nodi/modifica".

f3: Possibilità di cancellare un nodo se toccato dallo schermo, quando il software è in modalità “cancellazione”.

f4: Possibilità di selezionare un nodo se toccato dallo schermo, quando il software è in modalità “Selezione/inserimento archi”.

f5: Un nodo selezionato dovrà essere di colore diverso rispetto a quelli non selezionati, per distinguersi visivamente.

f6: Possibilità di applicare una trasform (traslazione, rotazione o scalatura) su un singolo nodo, tramite le rispettive gesture (slide, turn e pinch/stretch).

f7: La trasform applicata sul nodo scelto, dovrà essere replicata su tutti i nodi selezionati in simultanea.

f8: Possibilità di creare archi tra i nodi selezionati, toccando lo schermo in un punto dove non è presente un nodo, in modalità “Selezione/inserimento archi”.

f9: Lo smart desk e la lavagna LIM dovranno essere di fatto uno l’estensione dello schermo dell’altro. I 2 condivideranno il grafo tra i 2 schermi.

f10: Possibilità di “lanciare i nodi” tra uno schermo e l’altro. Un nodo su cui viene applicata una spinta dovuta a una gesture di traslazione, che esce da uno schermo dal verso attiguo al secondo, dovrà comparire in quest’ultimo mantenendo la spinta data dal lancio.

f11: Il software dovrà essere in grado di distinguere bene che tipo di gesture l’utente sta applicando. Bisogna quindi che il software capisca se un primo tocco è stato applicato con intenzione di creare un nodo o se è l’inizio di una gesture per manipolare uno di questi.

f12: Il software, essendo collaborativo, dovrà inviare messaggi per aggiornare il sistema o riceverli da quest’ultimo per rimanere al corrente della situazione del grafo sull’altro dispositivo.

f13: Un nodo che cambia schermo in seguito a una traslazione dovrà mantenere tutte le caratteristiche che aveva prima dello spostamento, relative a dimensione, rotazione e nodi collegati.

## 2.2 Requisiti non funzionali

f14: Il software, essendo collaborativo, deve prevedere ed evitare possibili problemi dovuti alla comunicazione. Esempio: creazione di 2 nodi in seguito a un singolo tocco, in seguito a messaggi errati, mancata sincronizzazione...

# Capitolo 3

# Design

Il progetto che si vuole realizzare, essendo un'app multitouch, deve impedire che ci siano ambiguità tra i tipi di input che si vogliono predisporre. In particolare, le gesture che si mettono a disposizione devono essere univoche e ben distinte dalle altre.

La possibilità di esercitare una pressione su un nodo o in un punto vuoto dello schermo con un dito, risulta calzante per funzionalità distinte; per evitare di effettuare con le dita movimenti meno intuitivi per avere una più vasta gamma di input, o rischiare ambiguità usando lo stesso tipo di input, il software sarà diviso in 3 possibili modalità (f1) per la modifica del grafo:

- Inserimento nodi/modifica: In questa modalità, toccando un punto vuoto dello schermo, si disegnerà un nodo nella posizione indicata.

- Selezione/inserimento archi: In questa modalità, toccando un nodo sullo schermo, questo diventerà selezionato se non lo era già, altrimenti verrà deselezionato. Toccando un punto senza elementi dello schermo si disegneranno archi tra tutti i nodi selezionati.

- Cancellazione: In questa modalità, toccando un nodo sullo schermo, questo verrà cancellato.

Verrà predisposto un pulsante virtuale, per permettere all'utente di cambiare modalità premendolo. La modalità di default, che sarà impostata appena avviato il software è "Inserimento nodi/modifica".

Toccando un punto dove non è presente nessun elemento, si disegnerà un nodo (f2), che avrà una dimensione standard prestabilita e nessun angolo di rotazione. Il nodo è stato deciso arbitrariamente che sarà di colore verde chiaro, con bordo nero.

In questa modalità sarà anche possibile manipolare i nodi, applicando su questi elementi le gesture (f6). Quindi con uno swipe si traslerà il nodo, con un pinch o uno stretch si scalerà e in fine con una turn si ruoterà.

Le modalità impediscono, che applicando una gesture, il tocco iniziale venga preso per un gesto di cancellazione o selezione. Cambiando modalità, la successiva che verrà impostata sarà "Selezione/inserimento archi".

Toccando un nodo che non è selezionato, questo entrerà formalmente in modalità selezionato (f4). Si distinguerà dagli altri visualmente, in quanto il nodo e il suo bordo diventeranno gialli (f5).

Viceversa, toccando un nodo che è selezionato, tornerà nello stato di deselezionato, sia formalmente che visualmente.

Nella modalità "Selezione/inserimento archi", quando viene applicata una gesture su un nodo, dovranno reagire anche tutti i nodi selezionati (f7), applicando le stesse transform del nodo originale.

Quindi, per esempio, se su un nodo applico uno stretch che risulta nel raddoppiamento della sua dimensione, anche i nodi selezionati dovranno essere ingigantiti del doppio.

L'ultima modalità rimasta, e la successiva che verrà impostata se premiamo il pulsante, è la modalità "Cancellazione".

In quest'ultima modalità, toccando un nodo, questo verrà cancellato (f3), togliendo qualsiasi riferimento a quest'ultimo e tutti gli archi a lui collegati.

Il software non è pensato per essere usato da un singolo utente, ma per essere collaborativo, in particolare è pensato per essere eseguito dallo smart desk e dalla LIM (f9) nel laboratorio "E" (campus di Cesena, sede distaccata UNIBO).

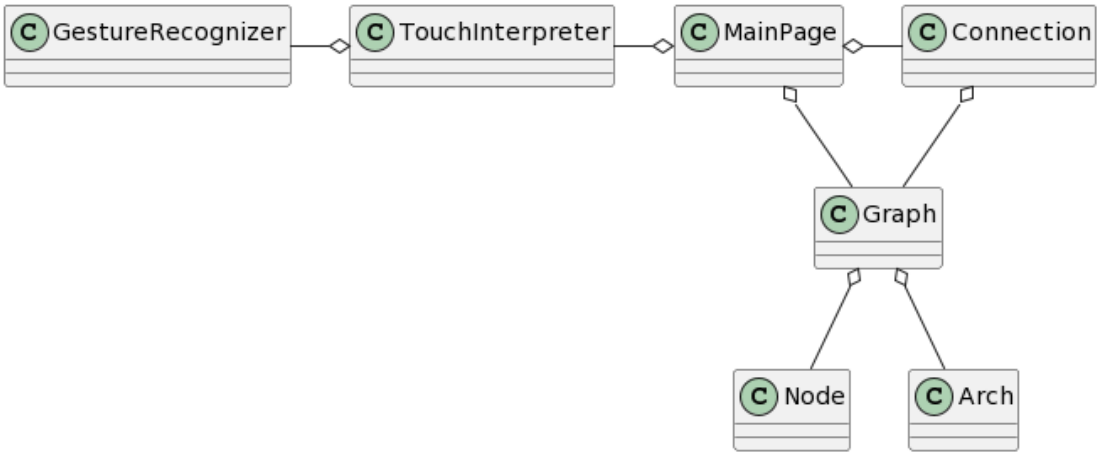
## 3.1 UML

Qui sotto riporto l'UML che descrive come saranno suddivisi i ruoli delle classi.

Sono ben distinguibili tre compiti principali di cui occuparsi:

- Disegno del grafo.
- Gestione della connessione.
- Riconoscimento e gestione delle Gesture.

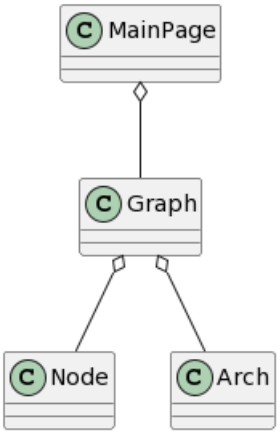
Figura 3.1: Il diagramma UML completo




Il grafo che l'utente dovrà essere in grado di disegnare, sarà formato da due elementi principali: il nodo e l'arco.

Ci aspettiamo quindi che ogni nodo memorizzi le proprie caratteristiche (dimensione, rotazione), mentre gli archi saranno identificati dai due nodi che collegano.

Figura 3.2: Il diagramma UML relativo alle classi per la creazione del grafo



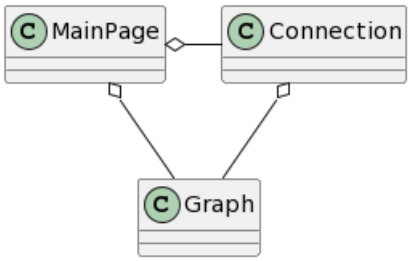

Gli elementi a schermo verranno disegnati in risposta agli input sul touchscreen che il programma sarà in grado di identificare.

A tale scopo, mi immagino che la gestione generale degli eventi "tocco" verranno affidati alla classe "TouchInterpreter". Qualora questa classe identificherà un tocco come possibile gesture, chiamerà in ausilio la classe GestureReocognizer, che si preoccuperà di riconoscere la gesture e gestirla.

Figura 3.3: Il diagramma UML relativo alla gestione del touch e delle gesture


La gestione della connessione e dello scambio di messaggi con il server, si occuperà principalmente di ricevere messaggi sulle modifiche del grafo apportate da altri client o di comunicare quelle applicate da se stesso. Sarà quindi necessario che la classe a cui viene assegnato questo compito, segnali tempestivamente al grafo come aggiornarsi.

Figura 3.4: Il diagramma UML relativo alla gestione della connessione



# Capitolo 4

# Implementazione e validazione

## 4.1 Implementazione del concetto di nodo

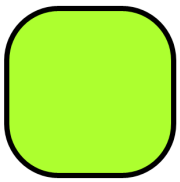
Listato 4.1: Disegnare il nodo su UWP

```
public NodeView(NodeModel nodeModel){
    ...
  this.rectangle.Fill = new SolidColorBrush(Windows.UI.
     Colors.GreenYellow);
  this.rectangle.Margin = thickness;
  this.rectangle.Width = nodeModel.getSize();
  this.rectangle.Height = nodeModel.getSize();
  this.rectangle.Stroke = new SolidColorBrush(Windows.UI
     .Colors.Black);
  this.rectangle.StrokeThickness = 3;
  this.rectangle.RadiusX = 30;
  this.rectangle.RadiusY = 30;
    ...
  }
```

Il concetto che va realizzato prima di implementare qualsiasi altra cosa è la classe "Node", che ho deciso di suddividire in una classe a disposizione per il modello e una per la parte visuale.

La classe "NodeModel" è importante che tenga conto della posizione in coordinate, della grandezza e della rotazione del nodo, nonchè se l'elemento si trova nello stato di selezionato o meno. La classe "NodeView" invece gestirà la parte visuale, disegnando il nodo e aggiornando graficamente le sue caratteristiche quando questa cambiano nel Model (esempio: allargare il nodo quando nel model il valore relativo alla dimensione aumenta).

Figura 4.1: un nodo disegnato dal codice sovrastante



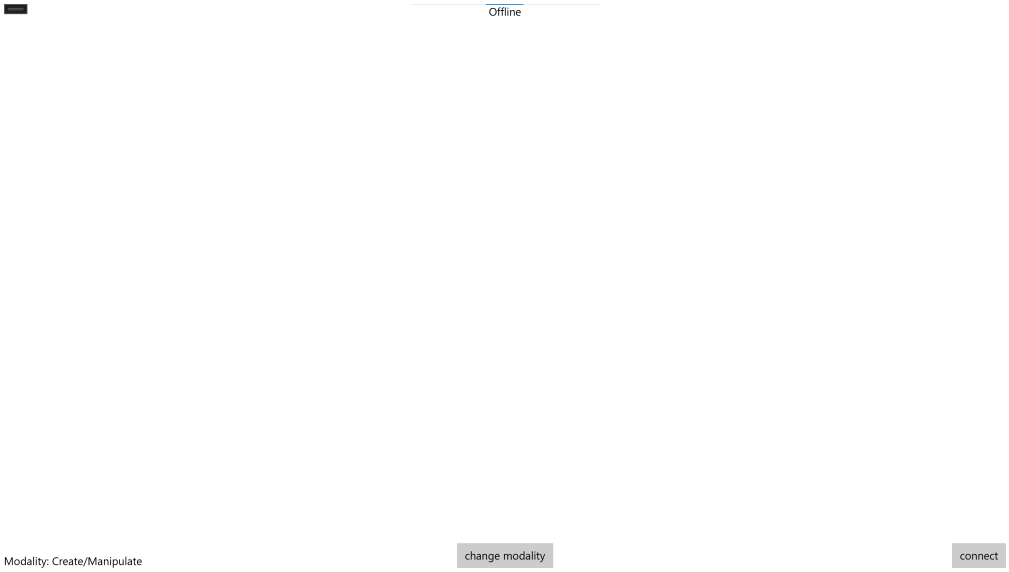
Quando il nodo entra in stato di selezionato, i suoi colori interno e del bordo verranno sostituiti con un giallo acceso, per far capire graficamente all'utente che l'elemento ha cambiato stato.

Listato 4.2: Cambio di colori al nodo per renderlo selezionato

```
1 void selectNode(NodeView node)
2 {
3   node.getRectangle().Stroke = new SolidColorBrush(
       Windows.UI.Colors.Yellow);
4   node.getRectangle().Fill = new SolidColorBrush(Windows
       .UI.Colors.Yellow);
5
6   ...
7 }
```

Figura 4.2: un nodo selezionato, colorato dal codice sovrastante

Figura 4.3: schermata principale senza aver inserito nessun elemento




## 4.2 Pagina principale

Graficamente, l'area di lavoro viene gestita dal linguaggio di markup XAML. UWP non permette l'implementazione di Canvas, quindi lo sfondo (e di conseguenza, l'inserimento di un elemento su di esso) verrà gestito da una griglia "Grid", che in questo progetto nomineremo "all".

Gli elementi che sono presenti in questa griglia appena avviato il software sono:

- Textblock "modalityText": mostra in che modalità di riconoscimento gesture è impostata in quel momento il software.
- pulsante "modButton": permette di cambiare modalità di riconoscimento delle gesture.
- pulsante "connectButton": permette di avviare un tentativo di connessione al server.
- Textblock "connectionText": mostra lo stato del tentativo di connesione.

Per come è stata creata la classe che si è deciso di utilizzare in questo progetto "Grid", non si possono disegnare su di essi elementi al di fuori del suo bordo in basso o a destra. La griglia "All" ha dimensione pari a quella dello schermo che si sta utilizzando, ma l'utente può disegnare elementi anche oltre questi limiti. Per questo motivo, la griglia conterrà al suo interno un'altra griglia più larga, nominata "Canvas" che permette all'utente di disegnare elementi fuori dallo schermo.

Listato 4.3: Finestra principale del progetto, in XAML

```
<Page
    ...

    <Grid Name="all" Background="{ThemeResource
       ApplicationPageBackgroundThemeBrush}">
        <Grid Name="canvas" Background="{ThemeResource
           ApplicationPageBackgroundThemeBrush}" Margin=
           "0,0,-2852,-2496">

        </Grid>
        <TextBlock Name="modalityText" Text="Modality:
           Create/Manipulate" VerticalAlignment="Bottom"
           ></TextBlock>
        <Button Name="modButton" Click="changeModality"
           ClickMode="Press" Command="{Binding
           WhateverCommand}" VerticalAlignment="Bottom"
           HorizontalAlignment="Center">change modality
           </Button>
        <Button Name="connectButton" Click="Connect"
           ClickMode="Press" Command="{Binding
           WhateverCommand}" VerticalAlignment="Bottom"
           HorizontalAlignment="Right" Width="69">
           connect</Button>
        <TextBlock Name="connectionText" Text="Offline"
           VerticalAlignment="Top" HorizontalAlignment="
           center"></TextBlock>
    </Grid>
</Page>
```

Da C#, invece, sono stati implementati i metodi che permettono all'utente di aggiungere, rimuovere o selezionare nodi, quando il software è in esecuzione.

Listato 4.4: Creazione e aggiunta di un nodo alla griglia

```
NodeView createNode(double x, double y)
{
  var (mappedX, mappedY) = mapping.fromView((x, y));
  NodeModel nodeModel = new NodeModel(mappedX, mappedY,
     sizeHeight);
  NodeView nodeView = new NodeView(nodeModel, nodes,
     bows, connection, nodesOutOfScreen);
```

```
6
7     nodeView.getRectangle().ManipulationMode =
          ManipulationModes.All;
8     grid.Children.Add(nodeView.getRectangle());
9     grid.Children.Add(nodeView.GetTextBlock());
10    nodes.Add(nodeView);
11    return nodeView;
12  }
```

Queste funzioni, dovranno rispondere a un input dato dalla gesture di tocco sul touchscreen, che, come detto in precedenza, in UWP corrisponde anche al click del mouse. Approfittando di questa tecnologia, è stato associato all'evento "PointerPressed", le funzioni di creazione, rimozione o selezione nodo, in base alla modalità in cui ci troviamo.

Listato 4.5: Funzione "OnTapped", scatenata dall'evento "PointerPressed"

```
1  private void OnTapped(object sender,
       PointerRoutedEventArgs e)
2  {
3    PointerPoint point = e.GetCurrentPoint(grid);
4    var x = point.Position.X;
5    var y = point.Position.Y;
6
7    var nodeClicked = findNodeClicked(x, y);
8    if (nodeClicked != null)
9    {
10     thisNodeExist = true;
11     this.startNode = nodeClicked;
12   }
13
14   if (!(this.thisNodeExist))
15   {
16     ...
17
18     var node = this.createNode(x, y);
19
20     ...
21   }
22   updateAllBows();
23   thisNodeExist = false;
24 }
```

Ultima funzionalità importante da citare in questa sezione, è il cambio di modalità di riconoscimento gesture da parte del software. L'evento della pressione del pulsante "modButton", scatenerà la funzione "buttonChangeModality", che cambierà la modalità in:

- "Select/Connect" se il software si trovava prima in "Create/Manipulate".
- "Delete" se il software si trovava prima in "Select/Connect".
- "Create/Manipulate" se il software si trovava prima in "Delete".

Per elencare le modalità disponibili si è scelto di fare uso di un enumeratore, che verrà associato al tipo di funzione da eseguire al tocco sul touchscreen.

Quindi a seconda di quale modalità sarà impostata in quel momento, la variabile "TouchModality" assume i valori:

- "Create_Manipulate"
- "Delete"
- "Select_Arch"

Listato 4.6: Funzione "OnTapped", scatenata dall'evento "PointerPressed"

```
private void buttonChangeModality()
{
  switch (touchModality)
  {
    case TouchModality.Create_Manipulate:
      grid.PointerPressed -= OnTapped;
      grid.PointerPressed += OnTapped_Select;
      touchModality = TouchModality.Select_Bow;
      modalityText.Text = "Modality: Select/Connect";
      break;

    case TouchModality.Select_Bow:
      grid.PointerPressed -= OnTapped_Select;
      grid.PointerPressed += OnTapped_Delete;
      touchModality = TouchModality.Delete;
      modalityText.Text = "Modality: Delete";
      break;

    case TouchModality.Delete:
```

```
20          grid.PointerPressed -= OnTapped_Delete;
21          grid.PointerPressed += OnTapped;
22          touchModality = TouchModality.Create_Manipulate;
23          modalityText.Text = "Modality: Create/Manipulate";
24          break;
25      }
26
27  }
```

Dai metodi "OnTapped", scopriamo uno dei punti deboli della tecnologia UWP; fatica a distinguere un tocco volontario, con l'intenzione di dare un input alla pressione, a una gesture applicata su un elemento. Questo fatto comporta che se in modalità "Create/Manipulate" l'utente voglia applicare una gesture "Turn" su un nodo, appena tocca l'elemento, il software, invece di comportarsi come aspettato, creerà un secondo nodo sopra quello già esistente.

Per ovviare il problema si è creata una classe "ClickDetector", che individua l'elemento più in evidenza toccato dall'utente e lo restituisce, impedendo che si generi un nodo se si applica la pressione in un punto dello schermo dove uno di questi è già presente.

## 4.3 Implementazione del concetto di Arco

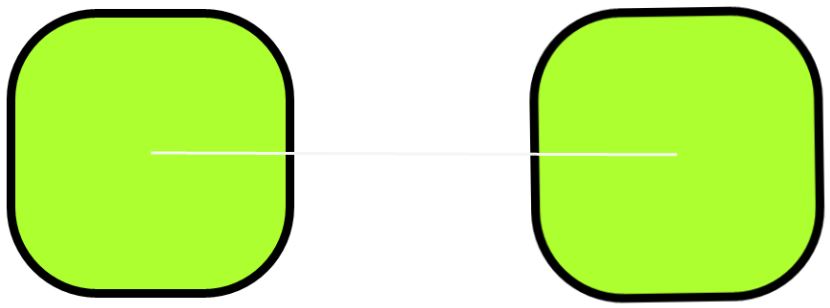
Come per il nodo, anche per l'arco si è deciso di implementare due classi diverse: "ArchModel" e "ArchView". Come è intuibile pensare, ArchModel si occuperà del modello dell'arco, registrando i dati relativi al nodo di partenza e al nodo di arrivo, mentre ArchView si preoccuperà di disegnarlo graficamente, rappresentato da una linea che collega due punti presi dal modello.

Listato 4.7: Creazione grafica di un arco, dal costruttore di "ArchView"

```
1   public ArchView(ArchModel archModel)
2  {
3    this.archModel = archModel;
4    this.mapping = ((App)Application.Current).Mapping;
5    line = new Line();
6    var (startX, startY) = mapping.fromModel((archModel.
       getStartNode().getX(), archModel.getStartNode().
       getY()));
7    var (endX, endY) = mapping.fromModel((archModel.
       getEndNode().getX(), archModel.getEndNode().getY())
       );
```

```
8
9     line.X1 = startX;
10    line.Y1 = startY;
11    line.X2 = endX;
12    line.Y2 = endY;
13
14    line.Stroke = new SolidColorBrush(Windows.UI.Colors.
          WhiteSmoke);
15    line.Fill = new SolidColorBrush(Windows.UI.Colors.
          WhiteSmoke);
16 }
```

Figura 4.4: Due nodi collegati da un arco



In questo software, gli archi vengono creati toccando un'area dello schermo dove non è presente un nodo, collegando tutti quelli selezionati tra loro, creando un nuovo cammino.

Il fatto che l'esistenza di un arco sia dipendente dall'esistenza di uno dei due nodi che collega, quando un nodo viene rimosso, tutti gli archi a lui collegati verranno rimossi.

Listato 4.8: Rimozione di un nodo e, quindi, dei relativi archi collegati

```
1 void deleteNode(NodeView node){
```

```
  for (int j = 0; j < archs.Count; j++)
  {
    if (archs.ElementAt(j).getBowModel().getStartNode().
       Equals(node.getNodeModel()))
    {
      grid.Children.Remove(archs.ElementAt(j).getLine())
         ;
      archs.Remove(archs.ElementAt(j));
      j--;
    }
    else if (archs.ElementAt(j).getBowModel().getEndNode
       ().Equals(node.getNodeModel()))
    {
      grid.Children.Remove(archs.ElementAt(j).getLine())
         ;
      archs.Remove(archs.ElementAt(j));
      j--;
    }
  }
  grid.Children.Remove(node.getRectangle());
  grid.Children.Remove(node.GetTextBlock());
  nodes.Remove(node);

  ...
}
```

## 4.4 Gesture di manipolazione

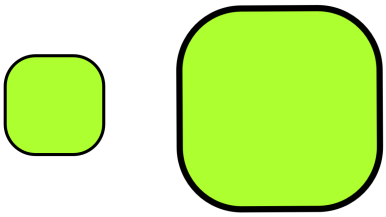
All'interno della classe "NodeView", vengono gestite le interazioni tra le gesture e il nodo. Un utente infatti può scegliere di spostare, ingrandire o ruotare un nodo a sua scelta.

Quando si effettua questo tipo di operazione, il software dovrà aggiornare la posizione anche di tutti gli archi collegati al nodo, qualsiasi tipo di trasformazione si stia effettuando e dovrà coordinarli in modo che questi continuino a puntare al centro dell'elemento.

Ai nodi soggetti da trasformazione viene inoltre applicato un movimento dato dalla classe "Inertia" di UWP che simula la fisica. Se infatti si trasla un nodo velocemente poi si toglie la pressione all'improvviso, l'elemento continuerà a muo-

versi nel verso in cui era diretto rallentando lentamente fino a fermarsi, simulando di essere stato lanciato.

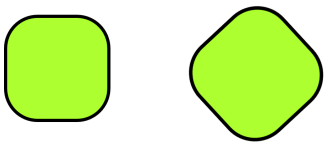
Figura 4.5: due nodi, quello a destra è stato soggetto ad una trasformazione di scalatura dovuta alla gesture di stretch



In questo progetto è previsto che quando a un nodo viene applicata una trasformazione, tutti i nodi selezionati debbano replicare la stessa, inerzia compresa. Quindi, se un nodo ad esempio viene spostato tramite una traslazione, tutti gli elementi selezionati si devono muovere nella stessa direzione dello stesso spazio.

L'implementazione di questa funzionalità causa una serie di problemi da gestire. In primis, se si applica la gesture su un elemento che è selezionato, bisogna impedire che la trasformazione venga applicata due volte ma solo una; infatti, senza gestire il problema, si trasformerà una prima volta perché riceve l'input della gesture e raddoppierà la dose per la sua caratteristica di essere selezionato. Bisogna quindi applicare la trasformazione a tutti i nodi selezionati, tralasciando il nodo che l'utente sta manipolando.

Figura 4.6: due nodi, quello a destra è stato soggetto ad una trasformazione di rotazione dovuta alla gesture di turn



Le trasformazioni di scalatura e rotazione sono dipendenti, molto più che la traslazione, dal punto di origine che si prende in considerazione. Infatti se applichiamo la scalatura a un rettangolo, prendendo come punto di origine della trasformazione il suo vertice in basso a sinistra, questo si allargherà nella direzione opposta, dando l'idea all'utente che il nodo gli stia "scivolando di mano". L'utente

invece si aspetterà probabilmente che il nodo si allarghi in tutte le direzioni e che quindi l'origine sia al centro dell'elemento. Per far si che questo accada bisogna esplicitarlo nel codice.

Listato 4.9: Impostare l'origine della trasformazione di scalatura al centro del nodo

```
private async void touchRectangle_ScaleDelta(object
    sender, ManipulationDeltaRoutedEventArgs e)
{
  Rectangle rect = sender as Rectangle;

  scaleTranslation.CenterX = rect.Width / 2;
  scaleTranslation.CenterY = rect.Width / 2;

  scaleTranslation.ScaleX += e.Delta.Expansion / 100;
  scaleTranslation.ScaleY += e.Delta.Expansion / 100;

  this.getNodeModel().updateSize(scaleTranslation.ScaleX
      * this.rectangle.Width);
  ...
}
```

Listato 4.10: Risposta alla gesture di swipe, tramite la trasformazione di traslazione

```
void touchRectangle_ManipulationDelta(object sender,
            ManipulationDeltaRoutedEventArgs e)
{
  // Move the rectangle.
  Rectangle rect = sender as Rectangle;

  dragTranslation.X += e.Delta.Translation.X/(this.
     getNodeModel().getSize()/100);
  dragTranslation.Y += e.Delta.Translation.Y/(this.
     getNodeModel().getSize() / 100);

  this.GetTextBlock().Text = dragTranslation.Y.ToString
     ();

  ...


```

```
15    for (int i = 0; i < archs.Count; i++)
16    {
17      archs.ElementAt(i).updateView();
18    }
19
20    for(int i=0; i < nodes.Count; i++)
21    {
22      var otherNode = nodes.ElementAt(i);
23      if (otherNode.getNodeModel().isSelect() && otherNode
          .getRectangle() != getRectangle())
24      {
25        otherNode.getDragTranslate().X += e.Delta.
            Translation.X;
26        otherNode.getDragTranslate().Y += e.Delta.
            Translation.Y;
27
28        otherNode.GetTextBlock().Text = dragTranslation.Y.
            ToString();
29      }
30    }
31  }
```

## 4.5 Comunicazione tra LIM e smart desk

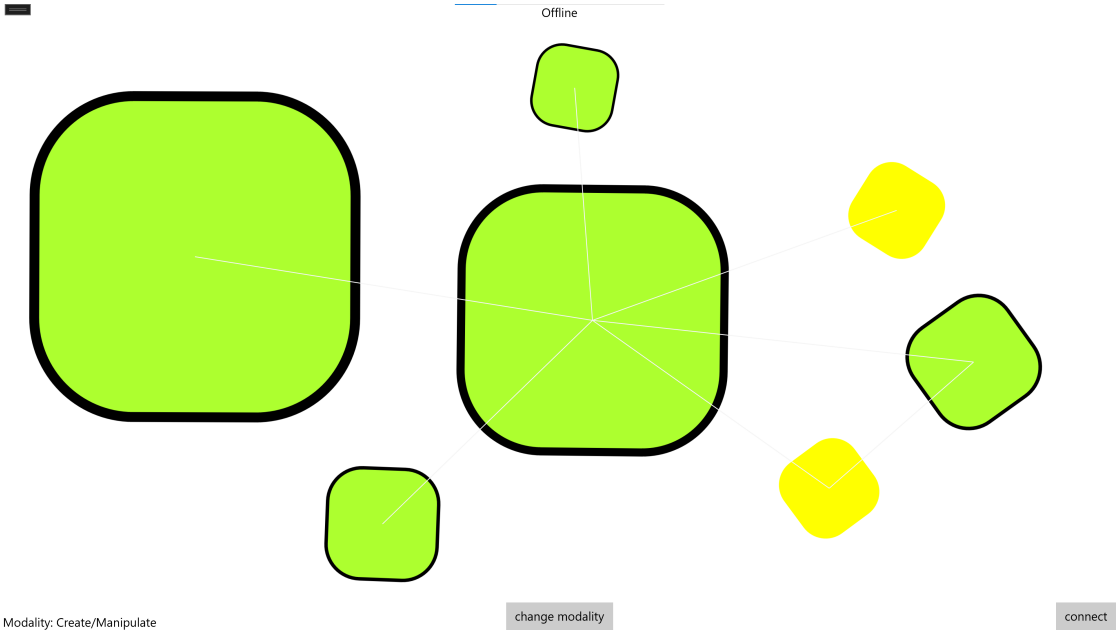
Come anticipato, lo smart desk dovrà essere di fatto l'estensione dello schermo della LIM. Essi condivideranno virtualmente lo stesso spazio di lavoro. Per permettere questo le coordinate del modello sono mappate a seconda di quale dispositivo si stia utilizzando tra i due. Se si usa la lavagna LIM, il punto in alto a sinistra viene considerato come l'origine del sistema e quindi corrisponderà in coordinate al punto (0, 0), mentre, contando che nei sistemi di riferimento cartesiano grafici generati da codice, il valore dell'asse Y aumenta quanto più ci si abbassa di quota, l'angolo alto a sinistra dello smart desk sarà identificato dalle coordinate (0, 1100).

Le coordinate dello schermo, lato view, sono indipendenti per ogni dispositivo. Si rende quindi necessario, a ogni messaggio che gestisce la creazione di un nodo, effettuare una conversione da coordinate del modello a coordinate visuali.

Per fare ciò, non si sono realizzati due software distinti (uno a dispositivo), ma l'applicativo riconosce il suo ruolo in base a una stringa dalla classe Application (a cui è possibile associare valori al suo campo Current da compilatore).

Listato 4.11: Sistema di mapping tra la View e il Model, a seconda del dispositivo

Figura 4.7: esempio di grafo, disegnabile tramite questa applicazione




```
1  public class CoordinateMapping
2  {
3    private double deltaY;
4    private double deltaX;
5
6    public CoordinateMapping(double deltaX, double deltaY)
7    {
8      this.deltaY = deltaY;
9      this.deltaX = deltaX;
10   }
11
12   public (Double, Double) fromView((Double, Double)
       coord)
13   {
14     return (coord.Item1 + deltaX, coord.Item2 + deltaY);
15   }
16
17   public (Double, Double) fromModel((Double, Double)
       coord)
```

```
18    {
19      return (coord.Item1 - deltaX, coord.Item2 - deltaY);
20    }
21  }
```

Il software che abbiamo descritto finora ha il ruolo di Client. Il server sarà un software distinto che si occuperà di gestire la connessione e di mettere a disposizione i metodi per lo scambio di messaggi che potranno richiamare i Client tramite RPC.

Una volta connessi, i Client richiamano il metodo "SendMessageAndCoordinate" del server, per comunicare tra di loro tutte le modifiche che vengono apportate al grafo e mantenersi sincronizzati.

La mappatura delle coordinate permette quindi che i due dispositivi vedano due aree distinte: la LIM vede tutti gli elementi con coordinate Y inferiore a 1100, mentre lo smartdesk vede gli archi e i nodi al di sopra di tale punto. Se viene creato un nodo, l'altro client dovrà generarlo nello stesso punto del modello; se la LIM crea un nodo nel punto (600, 200), anche lo smart desk farà altrettanto, aggiungendo il nodo in un'area non visibile dal proprio schermo, poichè il secondo vede gli elementi con coordinata Y superiore a 1100.

I nodi sono identificati da un id, per individuare correttamente l'elemento che sarà soggetto a modifiche in seguito al ricevimento di un messaggio dall'altro client. Ogni qual volta si effettua una trasformazione su un nodo, o questo viene selezionato o deselezionato, è necessario che il client informi l'altro dispositivo riguardo a tale modifica.

Listato 4.12: Ricezioni messaggi

```
1  private async void Connect(){
2    ...
3
4    connection.On<string, string, String, double, double,
        Boolean, double, double>("ReceiveMessage", (user,
        message, id, x, y, selected, size, rotation) =>
5    {
6      Windows.ApplicationModel.Core.CoreApplication.
          MainView.CoreWindow.Dispatcher.RunAsync(
          CoreDispatcherPriority.Normal,
7      async () =>
8      {
9        if (!user.Contains(desktopType))
10       {
11         if (message.Contains("add node"))
```

```
12          {
13            var (mappedX, mappedY) = this.mapping.
                 fromModel((x, y));
14            var node = this.createNode(mappedX, mappedY);
15            node.getNodeModel().setId(id);
16            if (selected)
17            {
18              selectNode(node);
19            }
20          }
21
22            ...
23
24        }
25      }
26      ).AsTask();
27    });
28
29    ...
30  }
```

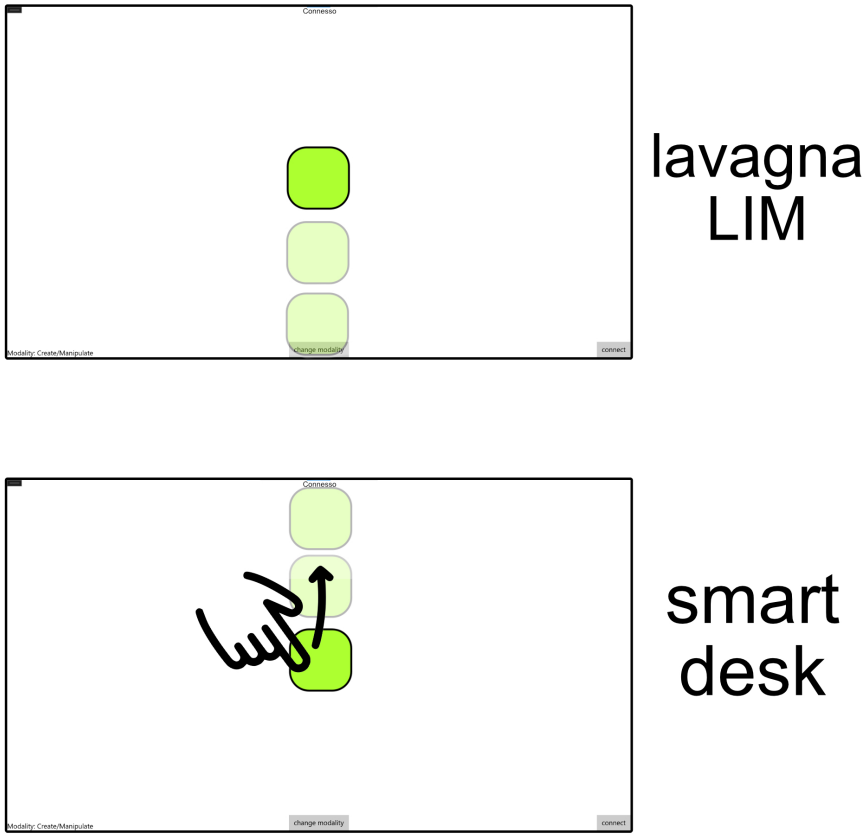
Visto che vengono individuati prendendo come punto di riferimento l'id del nodo, non è un problema la mole di messaggi che descrivono lo spostamento, in quanto, anche se si verificasse la remota ipotesi che un pacchetto non arrivi a destinazione, il suo successivo risincronizzerà il nodo con la posizione, dimensione e rotazione corretta.

Partendo dal presupposto che ogni client è aggiornato riguardo lo status di selezionato di ogni nodo di entrambi gli schermi, per creare un arco non sarà necessario inviare un messaggio per ognuno di essi, ma soltanto informare un'unica volta che vanno creati. Il software è infatti impostato per creare archi tra tutti i nodi selezionati presenti nel sistema.

Su ogni nodo selezionato, da entrambi gli schermi, si devono applicare le trasformazioni che si stanno verificando su un altro elemento tramite gesture di manipolazione. Se i nodi selezionati sono tanti, inviare un messaggio per ogni singolo elemento potrebbe richiedere una mole di messaggi non indifferente, che inciderebbe sulla prestazione del software. A questo punto si è scelto di mandare un messaggio univoco, che informi tutti i nodi selezionati di quanto muoversi e in che direzione. Nei capitoli dedicati alla comunicazione, è stato chiarito come la scelta di non inviare la posizione di destinazione in termini di coordinate, ma l'incremento di questa, possa causare un problema di mancata sincronizzazione dovuto alla possibile perdita di messaggi. In questo caso si è deciso di favorire le prestazioni,

a costo che alcuni nodi non siano allineati al millimetro. Va ricordato che appena si effettua una manipolazione direttamente tramite una gesture sul nodo, questo per forza di cose si risincronizzerà, ricevendo informazioni assolute riguardo alla sua posizione, dimensione e rotazione. Nei lavori futuri riguardante questo progetto, vedo necessario rivalutare la gestione della reazione alle trasformazioni di tutti gli elementi selezionati. Il futuro programmatore potrebbe riflettere su quale soluzione si dimostri maggiormente efficace, testando qual'è il limite per cui le prestazioni del programma vengono compromesse o se la mancata sincronizzazione momentanea genera incoerenze grafiche visibili.

Figura 4.8: la classe Inertia e la comunicazione traimte SignalR, permettono di lancira un nodo da uno schermo all'altro



# Capitolo 5

# Conclusioni

In questo progetto di tesi è stata effettuata un'esplorazione riguardante lo sviluppo di software collaborativi sfruttando schermi multitouch. Per approfondire, è stato sviluppato un prototipo che, nella fase attuale, soddisfa tutti i requisiti funzionali e non funzionali proposti.

Il programma dal lato della tecnologia multitouch si presenta solido, le avversità relative riscontrate durante la fase di implementazione sono state tutte risolte, ora sui nodi non ci sono problemi a distinguere che tipo di gesture si stia applicando ed effettuare le trasformazioni corrispondenti. I nodi selezionati reagiscono correttamente alle trasformazioni geometriche che vengono apportate sull'elemento manipolato da una gesture, sono infatti assenti problemi di sincronizzazione tra questi.

Sempre lato touch, il software presenta due pulsanti, uno per cambiare modalità e uno per avviare un tentativo di connessione. Una delle possibili modifiche che si potrebbero apportare, potrebbe essere sostituire i bottoni con eventi relativi alle gesture sullo sfondo, per rendere il programma completamente dedito a enfatizzare l'intenzione di voler incentivare l'utente a muoversi tramite tocco e movimenti coordinati della pressione delle dita sullo schermo.

Il lato della comunicazione, per il tempo limitato, non è stato approfondito come con le gesture. Lo scambio di messaggi risulta comunque efficace, quando i due software si connettono non si verificano problemi di mancata sincronizzazione e la mappatura tra grafica e la gestione del modello globale dei due dispositivi è ben gestita. Tuttavia, non si sono esplorate affondo tutte le problematiche, in quanto il server alla fine ha un ruolo di mero messaggero e non tiene traccia in maniera rilevante delle modifiche apportate al grafo. In sostanza, se i client si connettono, creano dei nodi o archi e uno dei due si disconnette per poi riconnettersi, il server non è stato progettato per informarlo riguardo allo stato attuale del grafo, portando a problemi di sincronizzazione tra i due terminali. Tra i lavori futuri, considero

essenziale rivalutare la gestione della comunicazione tra i client e il server, per dare al secondo una impronta più rilevante ed evitare questo tipo di errori.

Infine ci potrebbero essere diverse possibilità per arricchire questo programma; si potrebbe aggiungere una maggiore personalizzazione per i singoli nodi, permettendo di cambiare forma e colore, stesso discorso anche per gli archi. Si potrebbe pensare di permettere anche ad altri tipi di dispositivi di connettersi, per avere un ruolo che permetta di visualizzare l'intero grafo o "attaccarsi" ai due già presenti per aggiungere spazio di lavoro.

# Bibliografia


[1] Karl-Bridge-Microsoft, jwmsft, and hickeys, "Interazioni tramite tocco." link, 2022.

[2] S. Paganini, "Maniac i: il primo supercomputer in grado di battere l'uomo a scacchi (e costruire bombe atomiche)." link, 2023.

[3] G. Natalini, "Arpanet: la storia di internet prima di internet." link, 2019.

[4] M. Marcoaldi, "Che cos'è unix?." link, 2022.

[5] J. F. Knight, D. Deen-Williams, T. N. Arvanitis, C. Baber, S. Sotiriou, S. Anastopoulou, and M. Gargalakos, "Assessing the wearability of wearable computers," in *2006 10th IEEE International Symposium on Wearable Computers*, 2006.

[6] QuinnRadich, hickeys, SphinxKnight, cmcclister, and mattwojo, "Che cos'è un'app uwp (universal windows platform)?." link, 2022.

[7] A. Milli, "No, le uwp non sono morte, si stanno evolvendo." link, 2019.

[8] bradygaster, Rick-Anderson, alexbuckgit, guardrex, wadepickett, IEvangelist, yangzhongke, scottaddie, ahmad2smile, tdykstra, darthwalsh, and rachelappel, "Panoramica delle asp.net core signalr." ,link, 2023.