SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

Developing Extended Reality Real-Time Collaboration Methods for Spatial Design Processes

Felix Neumeyer

SCHOOL OF COMPUTATION, INFORMATION AND TECHNOLOGY - INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

Developing Extended Reality Real-Time Collaboration Methods for Spatial Design Processes

Entwicklung von Extended-Reality-Echtzeit-Kollaborationsmethoden für räumliche Designprozesse

Author:	Felix Neumeyer
Supervisor:	Prof. Gudrun Klinker, Ph.D.
Advisor:	Daniel Dyrda
Submission Date:	May 15, 2023

I confirm that this Master's Thesis in Informatics: Games Engineering is my own work and I have documented all sources and materials used.

Munich, May 15, 2023 Felix Neumeyer

Abstract

The integration of Extended Reality (XR) together with real-time collaboration is expected to enhance the efficiency and effectiveness of spatial design processes. In this thesis, a Virtual Reality-based spatial design prototype was created, incorporating a variety of collaborative tools and features such as multi-user undo/redo, shared assets and their placement, virtual pointing, and teleportation tools. A user study was conducted to assess the importance of these prototype features and to explore new ideas for collaborative tools. User study participants expect VR-based spatial design tools to effectively supplement existing workflows, particularly among those with previous VR experience. However they are not considered a full replacement for conventional 2D editor tools. Most user study participants expressed a desire to use a VR-based spatial design application.

Additionally, real-time collaboration was introduced to Hololayer, an existing industrial Augmented Reality (AR) system at Siemens. The integration of a WebSockets-based client in conjunction with a distributed key-value store enables real-time updates for entities within the Hololayer system. Individual properties are synchronized between the key-value store and the client-side using reactive design patterns. The integration lays the foundation for more complex collaborative features to be implemented for the Hololayer system.

Future work should consider integrating additional collaborative features into the spatial design prototype, guided by the suggestions from user study participants. The prototype could be used to evaluate collaborative VR-based spatial design methods in additional user studies. The Hololayer collaboration system could also be extended to enable additional functionality, like prohibiting multiple users to modify the same key-value pairs in parallel. Overall, this research highlights the transformative potential of XR technologies in enhancing the efficiency, effectiveness, and collaborative nature of spatial design processes.

Zusammenfassung

Die Integration von Extended Reality (XR) mit einem Fokus auf Echtzeit-Kollaboration in räumliche Designprozesse kann deren Effizienz und Effektivität steigern. In dieser Arbeit wurde ein Virtual Reality-basierter Prototyp für räumliche Designprozesse erstellt, der einige kollaborative Funktionen wie Mehrbenutzer-Undo/Redo, eine geteilte Asset-Bibliothek sowie die Platzierung von Assets, Laserpointer und Teleportation zu anderen Nutzern integriert. Eine Nutzerstudie wurde durchgeführt, um die Wichtigkeit dieser Prototyp-Funktionen für räumliche Designprozesse zu bewerten und weitere Ideen für kollaborative Werkzeuge zu sammeln. Insbesondere Teilnehmer der Benutzerstudie mit vorheriger VR-Erfahrung erwarten, dass VR-basierte Design-Funktionen bestehende Arbeitsabläufe effektiv ergänzen können. Sie werden jedoch nicht als vollständiger Ersatz für herkömmliche 2D-Editoren betrachtet. Die meisten Teilnehmer der Benutzerstudie würden eine VR-basierte Raumdesign-Anwendung verwenden wollen.

Zusätzlich wurde Echtzeit-Kollaboration in Hololayer integriert - ein industrielles Augmented-Reality-System von Siemens. Echtzeit-Aktualisierungen von Objekten im Hololayer-System wurden durch die Integration eines WebSockets-basiertem Clienten in Verbindung mit einer verteilten Schlüssel-Werte-Datenbank erreicht. Einzelne Attribute von Objekten werden von der Client-seitigen Anwendung unter Verwendung von reaktiven Programmiermustern mit der Schlüssel-Werte-Datenbank synchronisiert. Die Implementierung dient als Basis zur Entwicklung weiterer kollaborativer Funktionen für das Hololayer-System.

In Zukunft könnten weitere kollaborative Funktionen in den Prototypen für räumliche Designprozesse eingebunden werden. Dafür können die Vorschläge der Nutzerstudienteilnehmer in Betracht gezogen werden. Der Prototyp könnte außerdem in zukünftigen Nutzerstudien zur Evaluation kollaborativer Funktionen benutzt werden. Das Kollaborations-System für Hololayer könnte auch durch weitere Funktionalitäten ergänzt werden, wie beispielsweise das Blockieren von parallelen Schreibvorgängen auf dem selben Schlüssel-Werte-Paar durch verschiedene Nutzer. Zusammenfassend zeigt diese Arbeit das Potenzial von Extended Reality zur Steigerung von Effizienz und Effektivität von kollaborativen räumlichen Designprozessen.

Contents

1 Introduction

The rapid evolution of technology has significantly impacted the way we approach creative and design processes, with remote work having become an essential component across almost all industries. Virtual Reality (VR) emerged as a relatively cost-effective way to experience and leverage 3d environments, as VR headsets like the Meta Quest 2 are available for about 400€. Initially used for gaming and entertainment purposes, VR has potential to be a valuable tool in design-related disciplines, especially architecture and construction can benefit [1]. Spatial design tasks, such as level design for games and architectural design reviews may greatly benefit from collaboration within Virtual Reality as well. There are unique opportunities to enhance the efficiency, effectiveness, and overall quality of the design outcomes by allowing multiple stakeholders to work together in real-time, share insights, and streamline decision-making processes. Collaborative tools have already demonstrated their value in daily tasks, as evidenced by the widespread adoption of platforms like Google Docs[1], Microsoft Word[2], and Figma[3]. These tools facilitate seamless collaboration and teamwork, enabling users to work on the same documents, designs, or assets simultaneously, regardless of their location. Consequently, these tools have significantly improved the way individuals and teams approach their work and enhance productivity.

However, the potential of collaborative software for spatial design tasks in Extended Reality (XR), encompassing Augmented and Virtual Reality, remains relatively untapped. XR promises new opportunities for collaboration of designers and stakeholders. For instance, virtual reality can be used to enable full-scale architectural visualization of designs, which is impossible with traditional visualization techniques. In the context of level design for games, the benefits could be substantial. When developing games for VR, level designers often face the challenge of constantly switching between the traditional desktop-based game engine editor and the VR game view to evaluate the effects of their modifications. Furthermore, when multiple designers are working on the same scene from different PCs, changes by the other designer are not immediately visible. We aim to explore what kind of collaborative features are needed for effective spatial design within VR and AR, implement some of them in the context of a VR

[1] https://www.google.de/intl/en/docs/about/
[2] https://office.com/word/
[3] https://www.figma.com/

level design prototype and enable real-time collaboration for Hololayer - an industrial augmented reality system at Siemens that can also be used for spatial design tasks [2].

1.1 Research Questions

This thesis consists of two parts. In the first part, described in Chapter 3, the general goal was to develop a virtual reality level design prototype, including a set of collaborative features and tools. Additionally, we wanted to gather ideas for additional collaborative spatial design features and tools, rank them by their importance and evaluate the features of the prototype, by conducting a user study with mostly expert users. Those features are most probably not inherent to level design, but can be useful in various spatial design domains, including industrial augmented reality.

In the second part (Chapter 4) we are integrating a real-time collaboration approach into Hololayer - an existing industrial augmented reality system at Siemens. By implementing and integrating a WebSockets-based client into the Hololayer app we lay the foundation for additional collaborative features to be implemented for the Hololayer system.

The main research questions we want to answer are:

1. What kind of features are most important for collaborative spatial design?
2. Would level designers want to use VR for collaborative level design?
3. Is our approach for real-time communication in the Hololayer system viable?

1.2 Extended, Virtual and Augmented Reality

Extended Reality, Virtual Reality, and Augmented Reality are related but distinct concepts in the realm of immersive technologies. **Extended Reality** (XR) is an umbrella-term that encompasses all immersive technologies that combine the physical and digital worlds. It includes Virtual Reality (VR), Augmented Reality (AR), and Mixed Reality (MR), as well as future technologies that have yet to be developed. XR aims to create seamless interactions between users and the digital environment, enhancing various aspects of human experience.

Virtual Reality is a fully immersive, computer-generated environment that simulates physical presence in real or imagined worlds. Users wear headsets, like the Meta Quest, Oculus Rift or HTC Vive, which track their head movements and display 3D images, creating the illusion of being in a different environment. VR often incorporates other

sensory inputs, such as audio and haptic feedback, to provide a more convincing and engaging experience.

Augmented Reality involves overlaying digital information, images, or objects onto the real world, enhancing the user's perception of their surroundings. This technology can be experienced through various devices, such as smartphones, tablets, or specialized AR glasses like the Microsoft HoloLens. Unlike VR, AR allows users to maintain awareness of their physical environment while interacting with digital content, making it well-suited for applications like navigation, education and instructions.

1.3 From Spatial Design to Level Design in VR

Spatial design can be generally defined as any type of active spatial appropriation, whether it is a room or a landscape [3]. More specifically it refers to the design of human environments, encompassing aspects of interior design, architecture, urban planning, and landscape architecture [3]. Many of these aspects are also found in game level design, as its goal is to build convincing, immersive worlds for the player to experience while optimizing function, aesthetics or emotional response among other things. In addition to spatial design considerations, level designers also have to think about gameplay elements, modalities and abilities of the player inherent to the game they are building. Designing levels for virtual reality is also different from traditional platforms like PC or consoles, especially in comparison to games in third-person view. From talking with people who have worked with VR extensively I gathered that designing game levels for VR directly in VR could be of great benefit since you normally have to switch between the game engine editor and running the game on the VR headset. Being able to modify something directly in VR, or leaving annotations about what you want to change later in the engine editor can be very helpful.

2 Related Work

To my knowledge there is no academic work specifically about collaborative spatial design in VR. However there are applications for collaborative design in VR.

2.1 Collaborative XR and VR for Spatial Design

Commercial Products *Nvidia Holodeck* is a virtual reality collaboration platform that allows designers, coworkers and stakeholders to perform design reviews, focusing on realistic graphics [4]. From a demonstration video on the product page, it seems to feature tools like 3D pen annotations, a laser pointer, voice chat and the ability to change materials of objects [4]. Specific designs - buildings in the video - can be loaded onto a "HoloTable" as miniature models to discuss them and potentially enter them in their real life scale. However, the platform does not seem to have gained traction, as it is still in "Early Access" since its announcement in 2017.

Gravity Sketch[1] is a VR-based 3D design platform to create, collaborate and review. Its main purpose is creating 3d designs of objects that can be exported as 3D models. Creating assets for use in spatial design can be considered part of the spatial design process.

Tilt Brush[2] is a VR-based 3D painting application. Users can choose from various brushes for painting. It is more of an art rather than a design application, but 3d painting can also be considered a form of spatial design, or at least it can be a tool for spatial design.

The *Unity EditorXR Plugin* was designed to transform common functionalities from the traditional Editor into VR, like object transformations [5]. In addition, it also provides annotation tools like 3D drawings. However the EditorXR plugin is no longer developed by Unity and not supported in newer versions of the Unity Editor.

[1] https://www.gravitysketch.com/
[2] https://www.tiltbrush.com/

Academic Work Beever et al. developed a VR system and workflow called "LevelEd VR", which supports the level design and creation process for VR games [6]. Their system focuses on the blockout process and aims to improve the understanding of scale and object positioning in VR environments. Users of LevelEd VR reported that it allowed them to spend more time iterating on gameplay due to the improvements in judging scale and object positioning. The system also enabled seamless transitions between editing and testing, enhancing workflow efficiency [6]. The potential for efficiencies is also noted in terms of the system's runtime capabilities, which allow it to operate on standalone systems such as the Oculus Quest. This opens up the possibility of working away from a desktop or laptop PC, thereby making the design process more flexible and versatile. Comparing their runtime editor to other systems, such as Unity Editor XR and Unreal Engine VR Mode, was suggested as a future line of inquiry [6].

Continuing the investigation into different aspects of design in VR, Choi et al. [7] proposed a unique framework for designing user interfaces that provide architects with an alternative spatial environment in Virtual Reality (VR). The framework is based on the examination of three types of spaces: Intuitive Physical Space (IPS), Projected Digital Space (PDS), and Immersive Virtual Space (IVS). The IPS is the actual physical space in which we live and interact. The PDS is the digital representation of space projected onto a 2D screen, commonly used for plan drawings and bird-eye view renderings. Finally, IVS, which corresponds to VR, is an immersive 3D virtual space where designers can experience their creations, which is the main focus of the study [7]. Choi argues that IVS can combine the strengths of both the real physical space (IPS) and two-dimensional screen space (PDS). In IVS, designers can perceive the scale, proportion, and spatial relationships of their designs as they would in the real world (IPS). They can navigate around the design, get up close to specific parts, or move back to see the whole. In this way, IVS offers an intuitive understanding of space that is similar to our physical experiences in the real world [7]. While PDS excels at offering a broader perspective of the design, such as a bird-eye view or a plan view, IVS can replicate this by allowing users to zoom out or view their designs from an elevated perspective [7]. This can help in understanding the overall pattern or structure of a design [7].

By combining these aspects, Choi argues that IVS (Virtual Reality) provides a unique and powerful environment for the spatial design process. It allows architects to design in a virtual space that mimics the intuitive understanding of scale and space of the physical world (IPS), while also providing the broader perspective and precision of digital projections (PDS) [7].

It is also worth examining functionalities within collaborative systems that can en-

hance their utility. One such functionality is the ability to undo and redo actions. In multi-user applications this becomes a more complex task. Rendl et al. conducted an empirical study to understand user expectations regarding undo and redo functionalities in single and group work scenarios [8]. They discovered that groups generally expect a group-oriented undo function, while individuals expect a user-oriented undo. These expectations are often met through a regional undo functionality, which provides a global undo for each region where a group or individual is working [8]. Based on this insight, Rendl et al. presented the requirements for implementing regional undo in a large-scale multi-user application. They proposed four techniques for determining the workspaces or regions of different groups and individuals working simultaneously on the whiteboard. Besides a manual concept where users define regions themselves, they presented automated procedures that form regions based on document content, the position of users, or the users' field of view [8].

2.2 Networking

Ubi-interact is a framework for interactive applications combining individual distributed systems and devices over a network [9]. It discusses several design choices to make when developing such a system, including event-based communication, serialization and edge computing capabilities [9]. Its modularity allows users to integrate various systems and devices, already providing nodes for CSharp, Javascript, C++ and Java [9].

Netcode for GameObjects (Netcode) is a networking library specifically designed for Unity that allows you to simplify networking logic [10]. It enables the transmission of GameObjects and world data to multiple players simultaneously during a networking session. By using Netcode, implementation focus can be shifted from dealing with low-level protocols and networking structures to primarily creating your game [10]. We therefore use Netcode in our prototype implementation and describe it in detail in Section 3.3.

3 Collaborative Virtual Reality Prototype

This chapter describes the design and implementation of a collaborative spatial design prototype for editing scenes in VR inside the Unity Editor.

3.1 Key Use Cases

To come up with useful tools for collaborative spatial design, I thought about the tasks level designers have to fulfill and how they might want to collaborate with other people. I decided to write the use cases in the style of user stories. In software development and project management, a user story is an informal, natural language description of one or multiple features of a software system. It describes what a user wants to achieve while using the software, potentially also detailing why. I wrote down a number of user stories that describe common use cases for level designers in virtual reality, which are listed in the table on the next page.

I selected some features from the user stories that I thought were essential to collaborative level design. First and foremost, the prototype needed to support common base functionality, like locomotion and grabbing or transforming objects, but those are not collaborative features. From the list I selected the idea of adding assets from a predefined set of objects, since assets are what makes up most of a level. There also had to be a list of collaborators and the possibility to find them within the scene. In order to effectively talk about a level design task at hand, I also wanted to include the laser pointer and 3d sketch functionalities. Finally, I imagined that reverting or re-applying changes is extremely important to level design, as it is common to try out different object placements. The features I implemented did not necessarily have the highest importance rating on the user story list, but were still essential to the prototype in order to provide some base functionality. Other arguably more important features would have been more complex to implement, or would simply not provide the base functionality needed for the prototype.

Table 3.1: User Story Feature List

As a level designer in a collaborative level design session I want to...	**Implementation Details**	**Importance Rating (1-5)**
Know who is currently working on the level	Collaborator list	5
See where my collaboration partner currently is and travel to them	Mini map (miniature) with user icons; Add Teleport/Show on Map Button to Collaborator List	5
Direct my collaboration partner's attention to what I'm talking about	Laser pointer; Laser pointer that leaves a temporary trail where it hits; Highlight/outline shader	5
Undo/redo changes I made to objects	Collaborative undo/redo	5
Talk to my collaboration partners via voice chat	Integrate voice chat solution (Discord SDK?)	5
Import sketches or other images (png/jpg) to the level for reference	Import level sketch and align it with level dimensions, so you can add shapes during blockout phase on top of the sketch	5
Create change proposals with multiple variations to choose from, so that collaborators can decide which version they prefer	Notification icon with number of change proposals pending	5
Make save points between iterations with the ability to switch through different iterations collaboratively	For whole scene vs on a per-object basis; Tracking changes of each object and requiring acceptance of changes by either the author or a collaborator for it to be included in the iteration; History of changes; Ability to show a certain changed object's version as a preview (transparent?)	5

Continued on next page

Table 3.1 – *Continued from previous page*

As a level designer in a collaborative level design session I want to...	**Implementation Details**	**Importance Rating (1-5)**
Modify translation, rotation and scale of objects	Grabbing and releasing objects in arms reach; Transformation handles like in Unity Editor	5
Add assets from a predefined set of objects	Spawn prefabs that were defined in the Unity Editor	5
Create basic shapes (cubes, cylinders, spheres)	Built-in tools for creating and manipulating basic shapes	5
Measure distances and sizes together with my collaborators	Virtual shared ruler/measurement tape; Reference humanoid, unit cube...	4
Hand over an object to a collaborator	Natural interaction vs "sending" it over in front of the other person	4
Define a viewpoint from where the player is intended to see the scene	Can quickly teleport to the viewpoints defined for the scene; Have others look at the scene from a certain viewpoint; Have optional camera feed(s) from these viewpoints overlaid; Define an intended path along which the player is supposed to move; Animate camera along path	4
Annotate the level with free-form sketches (3d controller)	Drawing 3d lines; Arrows	4
Create free-form shapes	Using voxels or other built-in tools for creating and manipulating free-form shapes	4
Know when someone changed an object in the scene	Have notifications when an object is changed/added/deleted	3
Make a comment on a specific object to start a discussion and have others see it	List of unresolved comments/threads; Notification icon with number of unresolved comments	3

Continued on next page

Table 3.1 – *Continued from previous page*

As a level designer in a collaborative level design session I want to...	**Implementation Details**	**Importance Rating (1-5)**
Know who changed an object or made an annotation	Show author information for each object and annotation	3
See how the sunlight would shine from different angles or throughout the day	Ability to adjust time of day and see changes in lighting; Sun path diagram visualization	3
Annotate the level with text	Virtual keyboard is extremely cumbersome; Speech-to-text input; On devices like Quest Pro you could use pass-through to use keyboard	2
Annotate the level with icons	Predefined icons for typical game elements (enemies, items, quest interaction positions)	2

3.2 XR Interaction Toolkit

The XR Interaction Toolkit is a Unity package designed for creating both virtual reality (VR) and augmented reality (AR) experiences [11]. This high-level, component-oriented interaction system allows for 3D and user interface (UI) interactions derived from Unity input events [11]. Its framework primarily consists of base Interactor and Interactable components, as well as an Interaction Manager that integrates these elements [11].

The toolkit encompasses a range of components that cater to a variety of interaction tasks. These tasks include cross-platform XR controller input, which supports platforms such as Meta Quest (Oculus), OpenXR, and Windows Mixed Reality, among others [11]. It enables basic object interaction, such as hover, select, and grab functionalities [11]. Additionally, the toolkit facilitates haptic feedback through XR controllers and offers visual feedback, such as tint or line rendering, to highlight potential and active interactions [11].

The XR Interaction Toolkit also enables basic canvas UI interaction using XR controllers [11]. Moreover, it contains a utility for interacting with the XR Origin, a VR camera rig, which manages both stationary and room-scale VR experiences [11]. Overall, the toolkit provides the necessary tools for creating immersive and interactive XR experiences and base components for 3D interactions [11]. What it does not contain is a library of user interface components - it only allows interaction with the existing Unity UI canvas-based components, but since this prototype is focused on collaborative aspects, this is acceptable.

3.3 Networking (Netcode for GameObjects)

In order to implement collaborative features, I use Netcode for GameObjects (Netcode), which is a high-level networking library for Unity [10]. It enables sending GameObjects and their corresponding properties to other clients within a networking session, without having to implement low-level protocols and networking code. In this section I outline the main features and components of Netcode [10].

3.3.1 NetworkObject

Every `GameObject` that should replicate networked properties or send/receive Remote Procedure Calls (see below) must have a `NetworkObject` component attached to it. Each `NetworkObject` has a *NetworkObjectId* which uniquely identifies it among all clients.

Ownership

A `NetworkObject` is always owned by exactly one client at a time. Making changes to a *NetworkObject* usually requires Ownership. `NetworkObject` ownership can be transferred to any client by the server. This is often necessary when a client wants to modify an object's position. The client will have to request ownership of the `NetworkObject` in question first, before he's able to modify its position.

3.3.2 NetworkBehaviour

`NetworkBehaviour` is an abstract class that derives from `MonoBehaviour` and is used to create custom networked behaviours. Usually, to create networked objects, one can derive from `NetworkBehaviour` and use `NetworkVariables` and RPCs (see below) to synchronize state. Additionally to the `NetworkObject` component, a `GameObject` also requires at least one `NetworkBehaviour` component on itself in order to use synchronize networked state.

3.3.3 NetworkVariable<T>

`NetworkVariables` are a fundamental feature that allow developers to synchronize data across the network with minimal effort. It is a wrapper around standard data types, such as integers or strings, which automatically handles synchronization, ensuring that the data remains consistent across all clients and servers. NetworkVariables can be configured to use different update frequencies and to only send updates when the data has changed, optimizing network performance.

3.3.4 NetworkTransform

The `NetworkTransform` component is a `NetworkBehaviour` provided by Netcode, which synchronizes a `NetworkObject`'s transform (*position*, *rotation*, *scale*) between clients within the networking session. Since Netcode is server-authoritative, any change to a `NetworkObject`'s transform need to be issued by the host or server by default. In most use-cases of this prototype, i.e. grabbing, moving and rotating an object with the controller, changes to an object's transform should be immediate and not require the server to make the change first. Therefore, a `ClientNetworkTransform` component which inherits from `NetworkTransform` and overrides the `bool OnIsServerAuthoritative()` method while returning $false$, can be used instead, so that clients can make changes to a `NetworkTransform` immediately without requiring confirmation by the server. This ensures grabbing objects feels responsive without any networking-related lag. Owner-

ship of the `NetworkTransform`'s `NetworkObject` is still required to make changes to the `ClientNetworkTransform` if the client is not the host.

3.3.5 Remote Procedure Calls (RPCs)

RPCs are a critical feature of Netcode for GameObjects that allow developers to execute code on remote clients or servers. They enable communication between NetworkBehaviours on different machines, allowing for synchronization of actions and events. Netcode offers three types of RPCs: Server RPCs, which are called on the client and executed on the server; Client RPCs, which are called on the server and executed on the client; and Custom RPCs, which can be invoked and executed by any `NetworkBehaviour`.

3.3.6 NetworkManager

The `NetworkManager` handles the lifecycle of network connections, including establishing, maintaining, and closing connections between clients and servers. It also oversees the spawning and despawning of networked GameObjects. `NetworkManager` can be customized by extending the `NetworkManager` class and implementing custom connection logic, though this was not necessary for this project.

NetworkPrefabs

A NetworkPrefab is a prefab with a `NetworkObject` component attached, which allows it to be instantiated and synchronized across all connected clients. All prefabs that should get spawned during a networking session need to be added to the NetworkManager's NetworkPrefabs list. Only the server/host can spawn a NetworkPrefab. When a NetworkPrefab is instantiated on the server, the server sends a message to all connected clients, prompting them to instantiate the same prefab. This ensures that all instances of the prefab on the different clients are synchronized and have the same `NetworkObject` identifier, so that they can be addressed. Therefore, for this prototype every asset that can be added to the scene during a networking session has to be added to the NetworkPrefabs. I added an Editor script which makes it possible to automatically add all prefabs within a pre-defined folder, which contain a `NetworkObject` component, to the NetworkPrefabs list of the `NetworkManager`.

PlayerPrefab

A *PlayerPrefab* is a specific type of prefab in Netcode for GameObjects that represents the player-controlled objects in the scene. When a client connects to a server,

a *PlayerPrefab* is instantiated for that client, providing them with an avatar or representation within the collaborative level design environment. Each *PlayerPrefab* is automatically assigned to the corresponding client, ensuring that players have control over their own objects and can interact with the level as intended. The *PlayerPrefab* instance of every connected client can be easily accessed through the `NetworkManager`. For this prototype, the *PlayerPrefab* contains the whole XR-Setup of the user, like tracked headset and controllers and other user-specific objects. When a *PlayerPrefab* is spawned for other clients (in contrast to the local user's client), we need to deactivate all components that should only run on the *PlayerPrefab* instance of the local client. E.g. all *PlayerPrefab* instances which belong to remote clients have their head-pose and XR-controller tracking disabled as their transforms are driven by the remote client.

Recycle Network Ids

By default, the `NetworkManager` recycles the NetworkObject.NetworkObjectIds after a specified period of time. In case of this prototype, this had to be disabled in order to retain unique NetworkObjectIds if a `NetworkObject` had been despawned. This is due to the Undo/Redo feature described later, which allows to revert and then re-issue a spawn command, which causes the re-spawned object to obtain a new NetworkObjectId. A reference from the old to the new NetworkObjectId is saved, so that any undo/redo commands that reference the despawned object can still find the re-spawned object. If the old NetworkObjectId had been re-used, a different object than intended could be referenced inside an undo/redo command.

3.4 Tools and Features

This section describes the tools and features that were implemented for this prototype. Tools can be opened through the main menu by holding the right controllers primary button for one second. The main menu then opens in front of the controller, so that the different tools can be accessed quickly (see Figure 3.1).

3.4.1 Asset Menu and Placement

The Asset Menu allows a user to add various pre-defined assets to the scene. The assets are categorized based on the folder that contains them in the Unity project and shown in different tabs of the menu (see Figure 3.2). Users can scroll through the assets in each category by either hovering arrow buttons with their controller, or by holding the controller's trigger button while pulling the menu in the scroll direction. By pressing the arrow buttons, the scrolling speed increases. When selecting an asset with the

Figure 3.1: Left: User list with buttons for teleporting to other users. Right: Main menu with tool selection buttons for laser pointer, 3D pen, asset menu and measurement tool.

controller, it is spawned and attached to the `RayInteractor` of the controller. The object is spawned at a distance corresponding to the size of the object, so that large objects are spawned farther away, while small objects are held close initially. The distance to the controller is chosen based on the diameter of the asset's bounding box. This ensures that the asset is positioned far enough away in front of the user's right-hand controller, so that it is fully visible from the user's perspective, as spawning and holding a very large object, like a house, too close to oneself, feels uncomfortable. The `XRRayInteractor` in combination with the `CollabXRGrabInteractable` on each asset allows a user to rotate and move the object around before finally placing it. The menu closes when an asset is selected and spawned, but opens up again, when the object is placed, so that the next object can be selected quickly. To spawn multiple instances of the same asset without re-opening the Asset Menu, the "Repeated Placement" option in the menu can be used. The assets placed by users during the current collaboration session are added to the "Recent" tab of the asset palette, allowing users to quickly find them again and continue working where they left off.

Spawning Assets

Only the host/server can spawn *NetworkPrefabs* (see Section 3.3.6), therefore any client trying to spawn an asset needs to send a ServerRpc to request an asset to be spawned.

Figure 3.2: Left: Asset Menu with assets categorized in different tabs. Center: An asset that was selected through the menu is attached to the `RayInteractor` and being placed. Right: The rotation axis is visualized for the object being placed.

It passes the path/Asset Address to the prefab to be spawned, and a local request number to the ServerRpc. The client manages a *SpawnRequestDictionary* that maps the request number to a spawn callback. Once the server has spawned the asset, it will invoke a *ClientRpc* on the client who requested the asset to be spawned, passing a `NetworkObjectReference` for the spawned object and the request number to the client. The request number can be used to pass the spawned object to a spawn callback by finding the correct callback in the *SpawnRequestDictionary*.

3.4.2 Object Modification (`CollabXRGrabInteractable`)

The base functionality provided by the `XRGrabInteractable`, which is included in the XR Interaction Toolkit, is mostly sufficient, but there were a few issues. For example, rotating the grabbed objects would either depend on the orientation of the controller when the object was grabbed, or it would rotate the object only around one axis, but around the attach point, which lies at the intersection point of the `RayInteractor` and the object, on the outside hull of the object. This would make rotating objects awkward. Rather, I wanted the object to rotate around its own local coordinate system, as objects are usually centered. Additionally, I also added functionality to rotate the object around the object's local X, Y or Z axis. By pressing the joystick button, the user could switch between the axes to rotate around. When switching between the axes, the currently

selected axis is shown for a limited time (see Figure 3.2 on the right).

3.4.3 Laser Pointer

The laser pointer is an extension of the users's hands and arms and visualizes the pointing direction of the user if a button is pressed on the controller (see Figure 3.3). Using a Unity `LineRenderer` component, which is attached to the tracked 3D Controller, a line is rendered, starting at the user's controller position and extending until the line hits any object. The end of the line, meaning the line intersection with an object, is determined by using a Unity `Physics.RayCast`.

The laser pointer's main use is to enhance communication and interaction among team members. This allows users to easily and precisely indicate specific areas or objects within the environment, thus facilitating clearer and more efficient discussions about design decisions, potential modifications, or problem areas. This targeted pointing mechanism eliminates ambiguity and ensures that all team members are on the same page regarding the subject of discussion. It may also be used as a navigational aid, enabling users to guide each other in complex level designs. In addition to the pointing function, the laser pointer also measures the distance between the user's controller and the surface at which the laser hits. The distance is shown to the user as a small text box attached to the controller used for measurement. For other users the distance measurement is shown as an enlarged text box above the user performing the measurement as you can see in Figure 3.3.

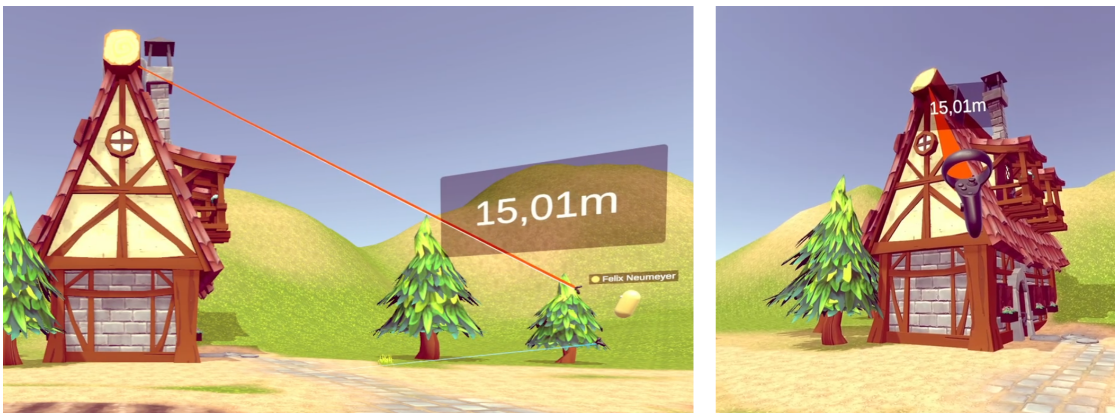

Figure 3.3: The user on the left observes the other user on the right, while he is pointing with the Laser Pointer. The distance measurement is shown to both users.

3.4.4 3D Pen

A pen has been implemented for making 3D sketches and annotations within the scene (see Figure 3.4). These sketches have a multiple potential use cases. They can be temporary representations of objects that will later be replaced with other assets. The 3D pen can be a valuable tool for enhancing the collaborative creative process. It allows level designers to quickly sketch out rough ideas, layouts, and spatial relationships directly within the virtual environment, enabling an intuitive and immersive design experience. The 3D pen can also be used to create annotations or labels directly on objects and surfaces, helping in communication among team members by providing context, explanations, or feedback on specific aspects of the design.

In addition to these functions, the 3D pen enables designers to experiment with various shapes and forms, quickly iterating and refining their ideas in 3D space. This flexibility facilitates rapid prototyping of level design elements, such as structures, terrain, and other assets, allowing designers to assess their feasibility and visual impact before committing to a final version.

One of the main advantages of using a 3D pen in a shared virtual environment is the ability to work together in real-time. Team members can sketch and annotate the design as they discuss and exchange ideas, fostering collaboration and streamlining the creative process. Furthermore, the 3D pen can help facilitate clear and efficient communication between designers, artists, and developers by allowing them to express their ideas visually and spatially, instead of relying solely on verbal or written descriptions.

Overall, incorporating a 3D pen into level design can significantly enhance the creative process by enabling intuitive sketching and annotation within the virtual environment, while promoting collaboration and efficient communication among team members. Collaborators don't have to possess artistic skills for using the 3D pen for basic annotations and to better communicate their thoughts.

3.4.5 Teleportation

The teleportation functionality can be accessed through the main menu to quickly find other users and travel to them. A user list is shown in the main menu with a Teleport button next to each connected user (see Figure 3.1). The algorithm tries to find a location next to the target user at a distance that is socially acceptable for colleagues at 1.8 meters away. If the user stands close to objects, the teleportation algorithm tries to spawn in the remaining free space. For example, when the sides (left and right) are blocked, the algorithm prioritizes the back of the target users, to not startle them or block their view. Finally, if there is absolutely no space around the target users, the user is teleported above the target user.

Figure 3.4: Left: Writing text in 3D. Right: Placeholder for a door asset to be added. The arrow indicates that the door should be functional.

3.5 Collaborative Undo/Redo

One of the most important features of any creative editing software is reversible actions, so users can undo/redo their changes. To quickly iterate, designers usually use undo functionality regularly. In level design, it also makes a lot of sense to compare different versions of object placements, as the scene can be observed from different viewpoints. While Undo/Redo can already be a tricky feature to implement on its own, it's even harder in a collaborative environment. Depending on the scenario, users might want to undo changes of other users, but in other cases, they only want to affect their own modifications. Therefore, there is the need for different Undo/Redo strategies. Below, we discuss the shortcomings of each method and how they may be mitigated by combination of methods. There are two different undo/redo methods implemented in this prototype (the Semi-Local and View-based methods) that the user could potentially switch between.

3.5.1 Undo/Redo Architecture

Reversible actions are implemented using the command pattern, which is the most well-known use for it [12]. Every action that needs an undo/redo, implements the `IModificationCommand` interface. In order to register commands on the server a *ServerRpc* needs to be sent. Every undo/redo action also has a corresponding *struct* that implements the `INetworkSerializable` interface by Netcode, which can be sent to

the corresponding ServerRpc. In order to make sure that state is always in sync between collaborators and since Unity Netcode for GameObjects already uses an authoritative server, all undo/redo commands are only stored and executed on the host/server. To issue an undo or redo operation, clients issue an undo or redo *ServerRpc*, which allows the server to find and execute the next command for the corresponding user. Depending on the strategy selected, the next undo/redo command will be determined with different algorithms.

3.5.2 Visualizing Undo/Redo Targets

Users have Undo and Redo buttons attached to their left hand controller. When they hover the buttons, the corresponding undo/redo target will be highlighted. The target object's shader is replaced to render it with a yellow tint instead (see Figure 3.5 on the left).

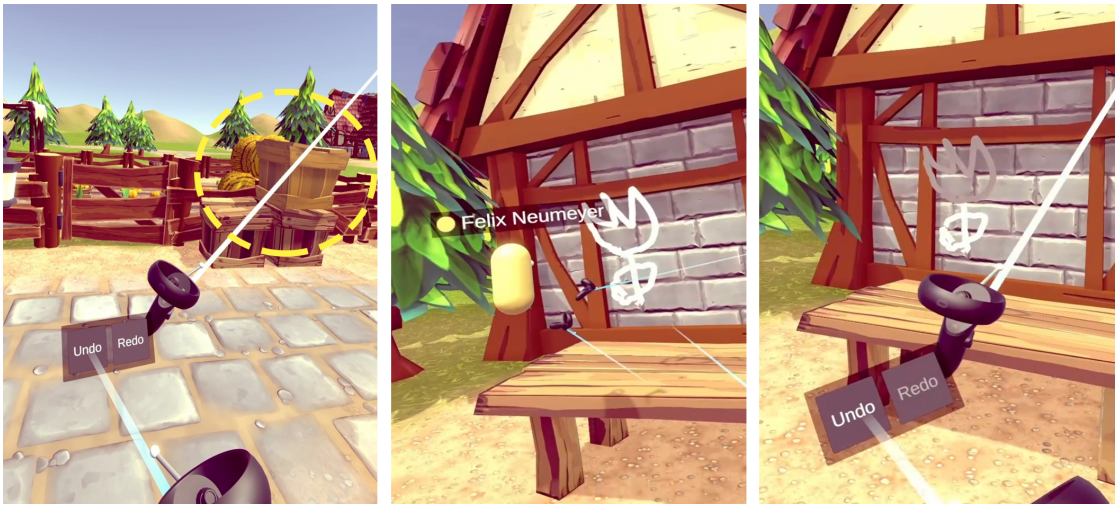

Figure 3.5: Left: Box on top is highlighted by a yellow tint as the next undo target. The yellow lines were added to clarify which box is highlighted, as it is hard to see in the screenshot. Center: Collaborator is drawing using the 3D Pen. Right: User is undoing the drawing made by the other user.

3.5.3 Undo/Redo Strategies

Global Undo/Redo

While this strategy was not used for the prototype, it's arguably the simplest strategy. For global undo/redo all modification commands by all users are stored in a sequential list of modifications, in the order they were applied. To undo or redo modifications, the list can simply be stepped through sequentially. If i is the index of the modification that was applied last, the next modification to undo and redo are located at indices $i-1$ and $i+1$ respectively. The implementation for collaborative global undo/redo is the same as for single-user applications, except that modifications of multiple users are stored in the modification list. The global undo/redo strategy is mostly not suited for collaborative environments, as users need to be able to undo their own changes without affecting changes made by other users. If an user A modifies object X, and user B afterwards modifies a different object Y, user A should ideally be able to undo the change to object X without affecting user B's change to object Y - but using the global undo/redo strategy, the change of user B to object Y has to be reverted first, before user A can revert the change on object X.

Local Undo/Redo

The local strategy only allows users to undo their own changes, but not changes by other users. It can be implemented by keeping a separate list of sequential modifications for each user, or by filtering a global list of modifications by the target user. As long as there are no dependencies between modifications made by different users, it is also fairly simple to implement. However, as the same object can be modified by multiple users, each user's modifications can not be applied or reverted sequentially without any further dependency checks. If user A modifies an object X, and then user B modifies object X, user A should not be able to revert his change on object X, as user B's change is the most recent change on object X. In order to undo the changes on object X, user B has to undo his change before user A.

Semi-Local Undo/Redo

To circumvent the short-comings of the local undo approach, that you can not apply or revert other users' changes, I propose the Semi-Local strategy. The Semi-Local approach allows users to revert changes to objects that were issued by other collaborators if their next local undo operation (which they issued themselves) would target this object. By allowing users to revert other people's changes on objects that they themselves have undo operations lined up for, users can efficiently work with objects that were collab-

orated on. However, there are still interactions between collaborators that wouldn't be handled well with this method. For example, if a user A moved an object X to the region of user B, where user B is currently working in, object X's by user A will not be available to user B by default, as he hasn't interacted with the object yet himself.

View-based Undo/Redo

Using the view-based method, users can undo changes on objects that they currently see, including objects modified by others (see Figure 3.5). This method was inspired by similar work that used a user's field of view for regional multi-user undo/redo for large interactive surfaces [8]. Users can select the region in which the next undo/redo operation will be selected by simply looking in the desired direction where their modifications to undo or redo are. We limit the objects to those inside the user's view, because we do not want to undo changes by other users that might be working elsewhere in the scene, as it would happen with the Global Undo/Redo strategy, but we still want to affect objects modified by others. Users may still encounter scenarios where they want to undo changes on an object within view, but there's another object right next to it with more recent changes. Repositioning to only have the wanted object within view might be cumbersome. Therefore we also propose the Object-based Undo/Redo strategy below, which was not implemented anymore due to time-constraints.

Implementation Details For undo/redo targets to be found within the view of the user, I implemented a method to cache all modifications within the view of the user that relies on Unity's existing physics implementations. Each command has a `PositionBefore` and `PositionAfter` property, optionally specifying where the command is positioned within the virtual environment. For each command, we add a GameObject containing a `SceneCollabModificationReference` component that references the modification command and has `SphereCollider` components at the PositionBefore and PositionAfter locations that are large enough to contain the target GameObject. As a child to the the Camera GameObject of a user, we attach a "Frustrum" GameObject. We add a `MeshCollider` and `Rigidbody` component (*isTrigger* and *isKinematic* are true) representing the camera frustrum to the *Frustrum* GameObject. On the same GameObject, the `FrustrumModificationReferenceCache` then keeps track of all GameObjects that collide with the frustrum collider and have a `SceneCollabModificationReference` component on them by overriding the OnTriggerEnter/Exit functions. Therefore, we can track all modifications that currently collide with the camera frustrum of each user. We sort the modifications by their $modificationId$ to find the newest/oldest commands.

An issue with the current implementation is that the GameObjects with

`SceneCollabModificationReference` components have SphereColliders and not MeshColliders that match their target objects exactly. That means that the Undo/Redo commands are not perfectly matching the positions of their target objects. A command targeting a large house for example has a very large SphereCollider that is larger than the house itself. This may cause the large object to be targeted even though it is not within view of the user. This can be mitigated by using the same collider as the target object, but would require to also store the rotation before/after a modification for each command. Therefore the objects within the `FrustrumModificationReferenceCache` do not perfectly match the objects that are visible to the user, which is not intuitive to the user.

Object-based Undo/Redo

The implementation of the View-based Undo/Redo already required the grouping of undo commands per object, as only the latest undo command of an should be considered as the next target command. We could allow users to access the history of changes of a specific object, e.g. by adding a Undo/Redo tool that uses the `RayInteractor` to select an object and listing all changes on that object. This may be preferable to the View-based Undo/Redo in some scenarios, when users want to affect just one object, but not everything within view.

3.5.4 Combining Methods

Since there is no one Undo/Redo Strategy that is perfect for every scenario, I assume that giving users the ability to select the Undo method they want to use is the best way to mitigate. Users may switch between methods depending on the current situation. For example, they may use the Local Undo/Redo method while working alone, but can switch to the View-based method when they encounter changes made by other users that they want to revert.

3.5.5 Undo/Redo Commands

Transform Modification The *TransformModificationCommand* allows applying and reverting of transform changes to an object. When a *CollabXRGrabInteractable* is grabbed and let go again, a new *TransformModificationCommand* is added to the *ModificationManager*, which saves the original and resulting transform (position, rotation and scale) from before grabbing and after repositioning the object.

Spawn The `SpawnCommand` can spawn prefabs that have been registered to the `NetworkManager`, deleted and respawned. This is achieved by providing the as-

set address as a string to the `SpawnCommand`. For redo to work on objects than have been despawned and respawned, we need to make sure that the `NetworkObjectReference` of the respawned object can be found. To realize this, I added a static `NetworkObjectReferenceExtensions` class that adds an extension method `GetRespawnedReference(...)` to the `NetworkObjectReference` struct. Every time an object is respawned through the `SpawnCommand`, we add an entry to a static Dictionary<NetworkObjectReference, NetworkObjectReference> within the Extension-class, with the old `NetworkObjectReference` as the key, and the new `NetworkObjectReference` as the value. This way we can always retrieve the latest `NetworkObjectReference` for respawned objects. All commands need to use the `GetRespawnedReference` method on their target `NetworkObjectReference` to make sure they can find respawned target objects.

Delete The delete command is the inverse of the spawn command. Internally, the delete command uses a spawn command. When executing the delete command, the internal spawn command is undone, and when the delete command is undone, the internal spawn command is executed.

3.6 Avatar and Personalization

Any collaborative VR environment needs an avatar to represent users within the system. For this prototype, we have only added a floating head built from Unity's integrated Capsule shape, with eyes that are Spheres. The hand positions are represented by floating VR controllers of the Meta Quest headset. Every user can set their user name and a color through a custom menu within the Unity Editor to allow rudimentary customization for their avatar.

4 Collaborative Industrial Augmented Reality Implementation

This chapter describes the implementation and integration of real-time communication capabilities using WebSockets into an existing industrial augmented reality system at Siemens, called Hololayer. This enables collaborative features such as the ones described in chapter 3 to be developed.

4.1 Hololayer

The main purpose of the Hololayer system is to provide a secure platform for creating, visualizing, interacting with, and persisting geo-referenced augmented reality (AR) content. The system allows users to create and edit holograms, which are digital 3D representations overlaying the real world, using a mobile app available on iOS, Android, and Hololens devices. Hololayer supports collaboration and sharing of AR content by structuring the world into Places, Layers, and Holograms, and by employing localization and spatial persistence techniques. The system is designed to be extensible and adaptable to various industrial and organizational use cases [2].

The Hololayer system is comprised of the following main components:

1. Hololayer server: Provides REST-based services for data storage, access, and management, as well as security features like authentication and authorization. It's a cloud-hosted back-end that stores and manages data related to holograms, places, and layers, and supports geo-spatial referencing and importing data from other sources like BIM (Building Information Modeling) or 3D asset management systems [2].

2. Mobile Hololayer AR client: Built with the Unity game engine and available on iOS, Android, and Hololens, this app allows users to create, visualize, and interact with holograms in the field. A modular architecture supports extensibility, easy maintenance, and integration with different devices and platforms. In this thesis, we add real-time collaboration to this client [2].

3. Hololayer Web UI: A web app for administrative tasks and user-interface management, supporting responsive design for use on various devices [2].

4. Data model: Structures the world into Places, Layers, and Holograms, enabling efficient organization, access, and collaboration on AR content [2].

5. Localization and spatial persistence: Combines GPS, image recognition, and SLAM (Simultaneous Localization and Mapping) techniques to achieve location accuracy, allowing holograms to be precisely positioned in the real world [2].

Hololayer is designed to address a variety of use cases across different contexts and industries. Some of these use cases include:

1. Remote assistance and collaboration: Hololayer enables experts to guide on-site personnel in real-time by overlaying holograms and visual instructions on the physical environment. This helps in reducing downtime and increasing efficiency [2].

2. Training and education: Hololayer can be used to create immersive training scenarios, allowing workers to learn new tasks and procedures in a safe, virtual environment [2].

3. Maintenance and repair: Hololayer can provide step-by-step instructions for maintenance and repair tasks, overlaying relevant information on the physical equipment, reducing errors and improving safety [2].

4. Planning and design: Hololayer can be used to validate and visualize plans in the design phase, enabling stakeholders to check the fit and collision of 3D models with the physical assets, and making adjustments as necessary [2].

4.1.1 Data Model

Holograms These are the virtual objects that users can create, visualize, and interact with in the Hololayer system [2]. Holograms can be of various types, such as 3D models, videos, images, or text. Metadata can be associated with holograms, including severity (error/warning/info), textual comments, or editing history [2]. Holograms are organized both logically and spatially to support efficient operations and wide-area data access [2].

Places Hololayer divides the world spatially into Places, which are usually around 10 meters in diameter, roughly the size of a typical room, but can vary in size [2]. A Hologram belongs to exactly one Place. Each Place has a GPS-referenced geographic location (latitude, longitude, altitude) and may have identifiers, such as a room number or a relatively unique object, like a piano, to distinguish it from other nearby Places [2]. Each Place also has a human-readable name and a preview image [2].

Localization within a Place in the Hololayer system involves several techniques to accurately determine the user's position and orientation [2]. Initially, the app uses GPS to get the user's approximate position and queries the backend for nearby Places [2]. The user then takes a photograph of a unique object to refine the list of potential Places [2].

Once the Place is identified, the app creates a SLAM (Simultaneous Localization and Mapping) map using natively supported APIs, which is persisted in the backend and associated with the Place [2]. This allows the app to recognize the position and orientation of the mobile device with an accuracy of around 20 cm in most environments [2].

In challenging environments, a Place may have Anchors with a position and an identifier [2]. If the app cannot relocalize itself based on the SLAM map, the user can point the phone at three or more anchors to re-localize accurately and update the SLAM map on the server [2].

Layers The world is structured logically into Layers in Hololayer, which can be displayed and edited individually. Each hologram belongs to exactly one Layer. Layers are used to categorize and organize holograms based on their purpose or function. Examples of Layers include "building maintenance," "electrical installation," or "tourist information" [2].

4.1.2 Authorization

Access to Places and Layers is managed using a fine-grained access control mechanism that assigns read, write, or admin access levels to each user for specific Places and Layers. This authorization scheme ensures that information is accessible only to authorized users, enabling them to create, edit, and visualize a given set of Holograms within the designated Places and Layers, while maintaining secure access and data management [2].

4.1.3 Existing Approach to Data Updates

Places, layers and holograms are fetched from the Hololayer REST-API. This means that clients that simultaneously modify holograms, do not immediately see changes made by other users. Each time a user opens the list of places or layers, a request to the REST-API is made to update the respective local data repository. Changes to Holograms only become visible to other users when they reload all holograms from the backend. While this already enables asynchronous collaboration between multiple users, it is not an ideal solution, as users can not see changes immediately and can not collaborate smoothly in real-time. One of the main goals of this thesis is to enable real-time collaboration between multiple clients by facilitating the collaboration server (see the following Section 4.2).

4.2 Collaboration Server

4.2.1 Overview

The Collaboration Server is a key-value store based on LevelDB[1], designed for distributed applications and accessible via WebSockets. It is a prototypical industrial approach to distributed computing that was implemented at Siemens prior to my thesis. The purpose of the Collaboration Server is to allow real-time state updates within the Hololayer system. Rather than using the Hololayer server's REST-API to fetch or modify data, clients now have the option to use the Collaboration Server's operations to modify the Hologram state. Once modified, the new state is automatically distributed to all other active clients that are currently connected. It offers an API with four streaming operations: sync (a database dump), put (adding or updating key-value pairs), del (deleting key-value pairs), and batch (atomically committing multiple operations), which allow clients to interact with the database. The server also handles access control, only allowing users with read privileges to connect, or users with write privileges to perform put or delete operations. A simple binary protocol to encode operations on the key-value store, ensuring efficient communication between server and clients. Server-side features include client ID generation for tracking, provenance and ownership management, and automatic clean-up of client-related data upon disconnection. This server offers a foundation for building various distributed applications while handling core functionalities related to the key-value store.

[1] https://github.com/google/leveldb

4.2.2 Collaboration Server Operations

The Collaboration Server supports four primary operations: sync, put, del, and batch. These operations are accessible via WebSockets.

put Operation

The *put* operation is used to add or update a key-value pair in the database. It consists of four parts:

1. The first byte encodes the message type (PUT).
2. The next four bytes encode the length of the key.
3. The following four bytes encode the length of the value.
4. The remaining bytes consist of the key and value byte arrays.

del Operation

The *del* operation is used to delete a key-value pair from the database based on a given key. It consists of two parts:

1. The first byte encodes the message type (DEL).
2. The next four bytes encode the length of the key.
3. The remaining bytes represent the key byte array.

batch Operation

The *batch* operation allows multiple put and del operations to be committed atomically. The structure of the message depends on the composition of the batched operations. The message can be decoded recursively as follows:

1. The first byte encodes the message type (BATCH).
2. The next four bytes encode the number of operations in the batch.
3. Each subsequent operation is processed serially, decoding the payload based on its respective message type (PUT or DEL).

sync Operation

The *sync* operation is the first message a client receives upon connecting to the Collaboration Server. It sends a dump of the current contents of the database as a batch message, allowing the client to synchronize its state with the server.

4.2.3 Storing Hologram State

As described above, the Collaboration Server is a key-value store. For storing Hologram state, each Hologram property can be saved as one key-value pair. For example, the position and rotation properties of a Hologram with the GUID `"550e8400-e29b-11d4-a716-446655440000"` are stored in the keys

`Holograms{0x00}550e8400-e29b-11d4-a716-446655440000{0x00}position` *and*
`Holograms{0x00}550e8400-e29b-11d4-a716-446655440000{0x00}rotation`

respectively. A zero-byte {0x00} is used for seperating the relation name (Hologram), Hologram GUID and property names (position and rotation). Clients can add or update these key-value pairs by invoking put operations with the respective key and value over a WebSocket connection to the Collaboration Server (see Section 4.2.2). This way, all modified Hologram properties can be stored, shared with other connected clients and finally persisted. The keys and values stored in the in key-value store can be arbitrary byte arrays, allowing one to serialize and store anything as a value.

4.2.4 Committing Hologram State

After changes have been made to one or more Holograms, the Collaboration Server contains a range of key-value pairs that are due to be either persisted through the REST-API backend, or discarded. Changes can be discarded by invoking delete operations on the Collaboration Server. To persist state, a commit request can be sent to the collaboration server. By default, the commit will persist all changes that are currently available in the key-value store. Alternatively, the range specifiers "lt" (less than) and "gt" (greater than) can be included in the commit request, allowing selective commits of the specified range of keys. For example, all states of a specific Hologram can be committed by specifying

$gt =$ `Hologram{0x00}{hologramId}{0x00}` *and*
$lt =$ `Hologram{0x00}{hologramId}{0x01}`

as all of the Hologram's property keys will fall between gt and lt. The LevelDB key-value store keeps its data sorted by key, allowing for fast iteration over all properties of a specific Hologram in order to commit them.

4.2.5 Scaling and Partitioning

The Collaboration Server was built with the Place and Layer model of Hololayer in mind and is part of a distributed system. At the time of writing this thesis, one instance of the Collaboration Server is supposed to handle exactly one Place/Layer combination. In the future, there may be one instance per Place that handles all Layers. Since Places already spatially partition the Hololayer system into small chunks, one Collaboration Server instance only has to handle a fairly small amount of connected clients, allowing the whole Hololayer system to scale horizontally. After multiple Holograms within a Place have been modified, those changes can be persisted at once through the Collaboration Server. By caching changes to Holograms on the Collaboration Server instances rather than immediately saving them through REST-API POST requests from each client, load is reduced on the REST-API and distributed to the Collaboration server instances. A reverse-proxy forwards connecting clients to the Collaboration Server instance for the Place provided with the connection request. If no Collaboration Server is running for the requested Place, a new instance can be spun up in the cloud before forwarding the client connection.

4.2.6 Cloud vs Edge Node

An edge node, or edge server, is a server that for example is physically located at the edge of a local area network. The Collaboration Server may be located on facility grounds as an edge node, so clients can connect to it through the local area network, allowing stable, low-latency connections. Since each Collaboration Server instance handles the load of exactly one Place, physically moving the server to where the Place is located is a reasonable choice, also allowing clients to stay connected to the local server instance if internet access within the facility is restricted or unavailable. The downside to edge servers is mostly cost, as physical servers need to be maintained on-site. Alternatively, the Collaboration Server can also be deployed as a cloud service, which increases latency but makes deployment hassle-free for any location that offers stable internet access. During my thesis, the Collaboration Server was deployed on AWS EC2 instances. The latency could barely be noticed as long as a stable internet connection was available.

4.2.7 Choice of Communication Protocol

WebSockets The Collaboration Server uses WebSockets to facilitate real-time communication between multi-platform clients and the server. Since WebSockets are based on the TCP protocol, they are connection-based and ensure reliable, in-order data transmission, which is usually required in Hololayer. WebSockets are designed for real-time applications, making them suitable for transmitting transient data that requires immediate synchronization between clients and the server. The need for compatibility with web frontends and Unity clients make WebSockets a natural choice, as they are widely supported in modern web browsers and various WebSocket client libraries for other platforms, including .NET and Unity, exist.

On the other hand, there are some disadvantages to using WebSockets for transmitting transient data. Rapidly changing data transmitted over WebSockets may cause network congestion, particularly if many clients are sending and receiving data simultaneously. Handling numerous WebSocket connections with high-frequency updates can become resource-intensive for the server, potentially affecting its ability to scale. Additionally, transmitting large amounts of transient data in real-time can consume significant bandwidth, which might be a concern for clients with limited network resources. Lastly, WebSockets use the TCP protocol, which ensures reliable data transmission. However, in scenarios where low latency is more critical than reliability, such as real-time gaming, occasional data loss might be tolerable. In such cases, protocols like UDP can be more suitable, as they prioritize low latency over guaranteed delivery. Reliable data transmission may not always be necessary in Hololayer, but there is not too much need for ultra-low-latency data transmissions and it is acceptable to also transmit these over WebSockets (TCP).

Alternative: MQTT (Message Queuing Telemetry Transport) The Collaboration Server, while being inspired by MQTT, has been designed to offer certain guarantees that MQTT does not. Specifically, the Collaboration Server ensures an initial state dump to newly connected clients, followed by state updates, which could be attempted with MQTT by retaining all messages and subscribing to all topics on newly connected clients. Also, the Collaboration Server is designed to be lightweight and deployable across various platforms, such as local systems, edge nodes, and cloud services.

The MQTT protocol provides an approach utilizing a publish/subscribe (pub/sub) pattern, which could have been used for the Collaboration Server. This pattern significantly differs from the traditional client-server architecture by decoupling the client that sends a message (the publisher) from the client or clients that receive the messages (the subscribers) [13]. Instead of direct communication between clients, a third component known as the broker is introduced to filter and distribute messages. This approach

results in a decoupling in three dimensions: space, time, and synchronization [13]. In space decoupling, the publishers and subscribers do not need to know each other, eliminating the need for exchange of IP address and port [13]. Time decoupling ensures that the publisher and subscriber do not need to run at the same time [13]. Finally, synchronization decoupling means that operations on both components do not need to be interrupted during publishing or receiving [13].

One of the core benefits of the MQTT pub/sub model is its scalability, as operations on the broker can be highly parallelized, and messages can be processed in an event-driven manner [13]. The broker achieves message filtering through various options, including subject-based, content-based, and type-based filtering [13]. This effective filtering mechanism allows each subscriber to receive only the messages of interest [13]. In case of the Collaboration Server, we are spatially partitioning the server already and all connected clients are supposed to receive all messages from each Collaboration Server, with few exceptions.

Originally designed for the communication of IoT devices MQTT was not built with very high update rates in mind, but has been shown to work well for collaborative augmented reality systems, as it comes with clustering capabilities [14]. In the work by Pereira et. al, a single instance can quickly become CPU-bound at around 120 concurrent users, while running two instances of a pub-sub broker as a cluster on a single machine caused a 1Gbit/s network to become congested at around 180 concurrent users [14]. Note that they are streaming positions of each user with multiple key points for facial expressions [14].

4.3 Implementation

The main Hololayer client is built using the Unity Game Engine. It's a cross-platform project targeting iOS and Android for handheld AR and Microsoft's Hololens for headmounted AR. The Unity project is built on software engineering principles from "Clean Code", which generally allow for extendability and maintainability [2, 15]. Data repository classes are used for managing Hologram, Place and Layer data, providing functionalities to fetch, create, modify or delete the respective data objects. The `IHologramRepository` interface defines the public API of repositories containing Hologram data. The `RESTApiHologramRepository`, which implements the `IHologramRepository` interface, provides methods to fetch, create, modify or delete Holograms through the Hololayer backend's REST-API. Since the `IHologramRepository` interface is used throughout the application to modify Holograms or to subscribe to Hologram changes, I decided to add a `CollabHologramRepository`, which inherits from the `RestApiHologramRepository` to fetch all Holograms available through the

REST-API backend, but also publishes and receives state updates for Holograms in real-time using the Collaboration Server. Since the majority of the Hololayer client components are designed to react to changes in the data repositories, minimal additional work is required, aside from providing updated Holograms through the `CollabHologramRepository`.

4.3.1 UniRx, Subjects and Observables

This subsection provides an explanation of the UniRx library components and their roles in the implementation of the Hololayer client. Unity Reactive Extensions (UniRx) is a port of the Reactive Extensions (Rx) library for the Unity game engine [16]. Rx is designed to create asynchronous and event-driven applications by utilizing observable sequences and LINQ-style query operators. With Rx, developers can represent asynchronous data streams through Observables, manipulate and query these streams using LINQ operators, and manage concurrency within the streams by using Schedulers [17]. UniRx was already used throughout the Unity Hololayer client, but is great for networking implementations as well. The main components of UniRx that my networking implementation uses are `IObservable<T>`, `Subject<T>`, `BehaviourSubject<T>`, `ReactiveProperty<T>` and some of the operators like *Buffer*, *Sample*, *GroupBy*, *Merge*, *Where* and *Select* [16, 18]. Since Observables, Observers and Subjects are being used heavily throughout the implementation, they are described below. Observables were already used in the existing repository APIs to allow subscriptions to Place, Layer and Hologram data.

Observable (`IObservable<T>`) An Observable is a design pattern used in reactive programming that represents a data stream, which can emit multiple values over time. Observables enable the asynchronous processing of data and events by subscribing to them using Observers. They provide an efficient and composable way to handle asynchronous data flows, such as user interactions, network requests, or timer events.

In the UniRx library, an Observable is represented by the `IObservable<T>` interface, where T is the type of the data being emitted by the Observable. The `IObservable<T>` interface has a single method, Subscribe, which allows an Observer to subscribe to the Observable.

```
public interface IObservable<out T>
{
    IDisposable Subscribe(IObserver<T> observer);
}
```

Observer (`IObserver<T>`) An Observer is an object that subscribes to an Observable in order to receive and react to the values, completion, or error notifications emitted by the Observable. In UniRx, an Observer is typically implemented using the `IObserver<T>` interface, where T is the type of the data being emitted by the Observable. The `IObserver<T>` interface defines three methods that an Observer needs to implement:

```
public interface IObserver<in T>
{
    void OnCompleted();
    void OnError(Exception error);
    void OnNext(T value);
}
```

OnNext(T value) is called by the Observable when it emits a new value. The Observer can react to the value, for example, by updating the UI, processing data, or triggering some action.
OnError(Exception error) is called by the Observable when an error occurs during the data emission process. The Observer can handle the error, for example, by logging the error message or displaying an error notification to the user.
OnCompleted() is called by the Observable when it has finished emitting values and no further values will be emitted. The Observer can perform any necessary cleanup or finalization tasks upon receiving this notification.

Subject (`ISubject<T>`) A Subject is a special type of Observable that can also act as an Observer [18]. It's a hybrid object that allows you to both emit and receive values in a reactive stream. In UniRx, instances of the `Subject<T>` class can be subscribed to by Observers and items of type T can be emitted by calling the `OnNext(T)` method of the subject.

```
public interface ISubject<T> : IObserver<T>, IObservable<T>
{
}
```

4.3.2 Collaboration Components

This section describes the various classes and components I developed to enable real-time collaboration within the Hololayer client.

CollabKey The CollabKey class corresponds to a key stored on the Collaboration Server (see Section 4.2.3). It is a simple wrapper around the binary keys transmitted

by the collaboration server, which allows you to directly access the key's *Relation* and *Name* segments. In case of a Hologram, the `HologramCollabKey`, which inherits from `CollabKey`, splits the *Name* into *HologramId* and *PropertyName*. For example, the key

`Holograms{0x00}550e8400-e29b-11d4-a716-446655440000{0x00}position`

is split into the following three parts:

$$\begin{aligned} Relation &= "Hologram", \\ HologramId &= "\texttt{550e8400-e29b-11d4-a716-446655440000}", \\ PropertyName &= "position" \end{aligned}$$

When sending an operation to the collaboration server, the `CollabKey` needs to be converted to a byte array, which can be done using its `AsByteArray(): byte[]` function.

CollabPropertyMessage A `CollabPropertyMessage` references the `CollabKey` for which property it received the message, and the property value itself as a byte array. All websocket transmissions containing key-value pairs are transformed to `CollabPropertyMessage`, which on their own do not know the type of value they contain.

CollabPropertyRepository This class is responsible for synchronizing and managing the collection of properties stored as key-value pairs on the Collaboration Server. It takes care of registering and deregistering properties, handling property changes, and managing the communication of property updates between clients. It is also responsible for managing the connection state and handling incoming and outgoing messages, parsing them into `CollabPropertyMessage` instances.

CollabProperty<T> This generic class represents a collaborative property, which can hold any type of data for which an `ICollabBufferReaderWriter<T>` has been registered before (e.g., int, string, Vector3 or custom classes). See section 4.3.4 on serialization. It synchronizes the state of exactly one key-value pair on the Collaboration Server, which is identified by a *CollabKey* property. It inherits from UniRx' `ReactiveProperty<T>` to maintain a reactive state, allowing observers to react to changes in the property value. The class also handles remote changes to the property value by subscribing to the property messages provided by the *CollabPropertyRepository*. Local changes to the property also automatically sent to the *CollabPropertyRepository* in order for them to be synchronized with the Collaboration Server and thus other clients. A CollabProperty<T> instance is typically initialized using the corresponding persisted

state of the property, as provided by the Hololayer REST-API. This is the case for all Hologram properties, as described in the `CollabHologram` paragraph below. CollabProperty instances can also be used to synchronize any other key-value pair on the Collaboration Server. The general idea for the `CollabProperty<T>` class came from Unity's Netcode for GameObjects [10]. Netcode has a `NetworkVariable<T>` class that enables synchronization of object properties between multiple clients [10].

CollabHologram This class represents a `Hologram` that utilizes `CollabProperty<T>` instances to store and synchronize its properties, such as *position*, *rotation*, and *scale*, with other clients and the Collaboration Server. Every time a Hologram property changes, the `CollabHologram` in turn emits a new version of a `Hologram`, as the `CollabHologram` provides an `IObservable<Hologram>` stream.

CollabHologramRepository The CollabHologramRepository allows other parts of the application to fetch, create, modify and delete Holograms. For every Hologram available, it creates an instance of CollabHologram, which in turn can receive remote changes to the Hologram and push local Hologram changes to the Collaboration Server. Since the *CollabHologramRepository* implements the *IHologramRepository* interface, which has a *PersistHologramChanges*(...) method for persisting local Hologram modifications, this method can be used to send local changes to the Collaboration Server, instead of persisting them using the Hololayer server REST-API.

4.3.3 State Synchronization

In order to synchronize state of Holograms with the Collaboration Server, each Hologram's property is synchronized with the corresponding key-value pair on the Collaboration Server. The keys on the Collaboration Server are structured as described in Section 4.2.3. As the Hololayer Unity project makes heavy use of UniRx, we decided to handle the WebSocket messages with UniRx as well. UniRx is a great fit for handling network streams, as they are basically observable sequences, which UniRx is built to handle.

The incoming WebSocket stream of Collaboration Server operations (see Section 4.2.2) is parsed so that the different operations can be handled. They always contain the key which identifies the property transmitted, so the operations are grouped by the key using the UniRx *GroupBy* operator. The resulting Observable outputs a stream of `GroupedObservable<CollabKey, ICollabPropertyMessage>`, where each `GroupedObservable` corresponds to the stream of updates for one key-value pair on the Collaboration Server. As a result, each key-value pair on the Collaboration Server has a corresponding UniRx Observable. These Observables are stored in a

`Dictionary<string, IObservable<CollabPropertyMessage>>`, mapping keys to Observable message streams for each key.

4.3.4 Serialization of `CollabProperty<T>` Values

The serialization is based on Google Protocol Buffers (Protobuf) [19]. Protocol Buffers represent a flexible and efficient serialization format, designed to handle structured data across various programming languages and platforms. They are extensively used at Google for a range of purposes, including inter-server communications and persistent data storage. Data structures are defined using .proto files, which allow for the seamless serialization and deserialization of data [19].

The Protobuf compiler automatically generates code in multiple programming languages, facilitating convenient manipulation of the corresponding protocol buffer. This code generation provides simple accessors, as well as serialization and parsing methods for developers. Notably, Protocol Buffers support seamless changes, enabling the addition or removal of fields without negatively impacting existing services or necessitating code updates [19]. I based my choice of serialization format and Protobuf message definitions on the work by Sandro et al., who discuss several options for serialization [9].

Central to the serialization of `CollabProperty<T>` values is the `CollabBufferReaderWriterLocator` class, which serves as a registry for associating the generic data types with their corresponding reader-writer class. The `CollabProperty<T>` class automatically serializes its value of type T using the appropriate serializer, which it can find using the `CollabBufferReaderWriterLocator` class. The serializers are classes that implement the `ICollabBufferReaderWriter<T>` interface, which defines a `void Read(byte[] buffer)` method for deserialization of a byte array into the target type and the corresponding `byte[] Write(T value)` method for serializing values from the target type into a byte array.

Each `ICollabBufferReaderWriter<T>` implementation will use the corresponding generated Protobuf code for serialization. There can be types that do not use a Protobuf serialization, as each `ICollabBufferReaderWriter<T>` implementation can use a different way of serializing values. For example, the *string*, *Vector*3, and *Quaternion* types still use a legacy serialization method that was used before the start of my thesis.

An overview of the classes and interfaces involved in the serialization process can be seen in the class diagram in Figure 4.2.

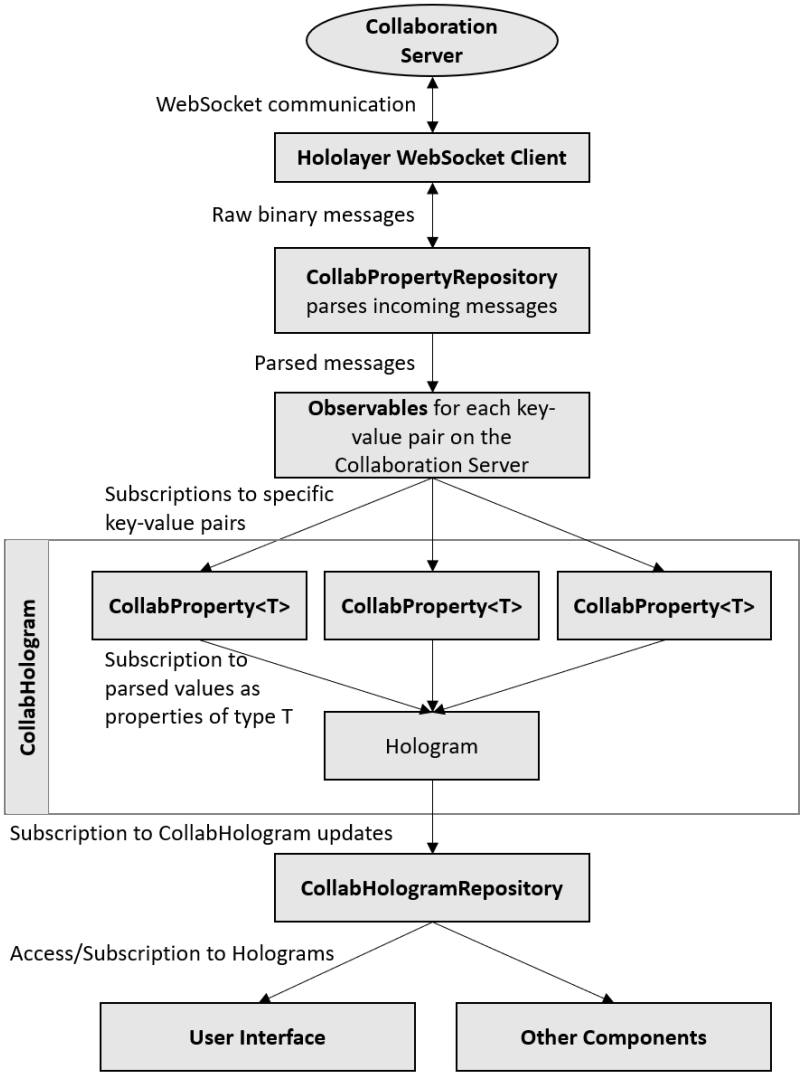

Figure 4.1: Data flow of incoming messages that are parsed into their key-value pairs and emitted in Observables for each key. CollabProperties inside CollabHologram instances subscribe to specific key-value pairs, like for example the position key ("Hologram{0x00}{HologramId}{0x00}position") and modify their internal Hologram instances that in turn are subscribed to by the CollabHologramRepository, which is keeping a collection of CollabHolograms. Other components can then access Hologram data through the CollabHologramRepository.

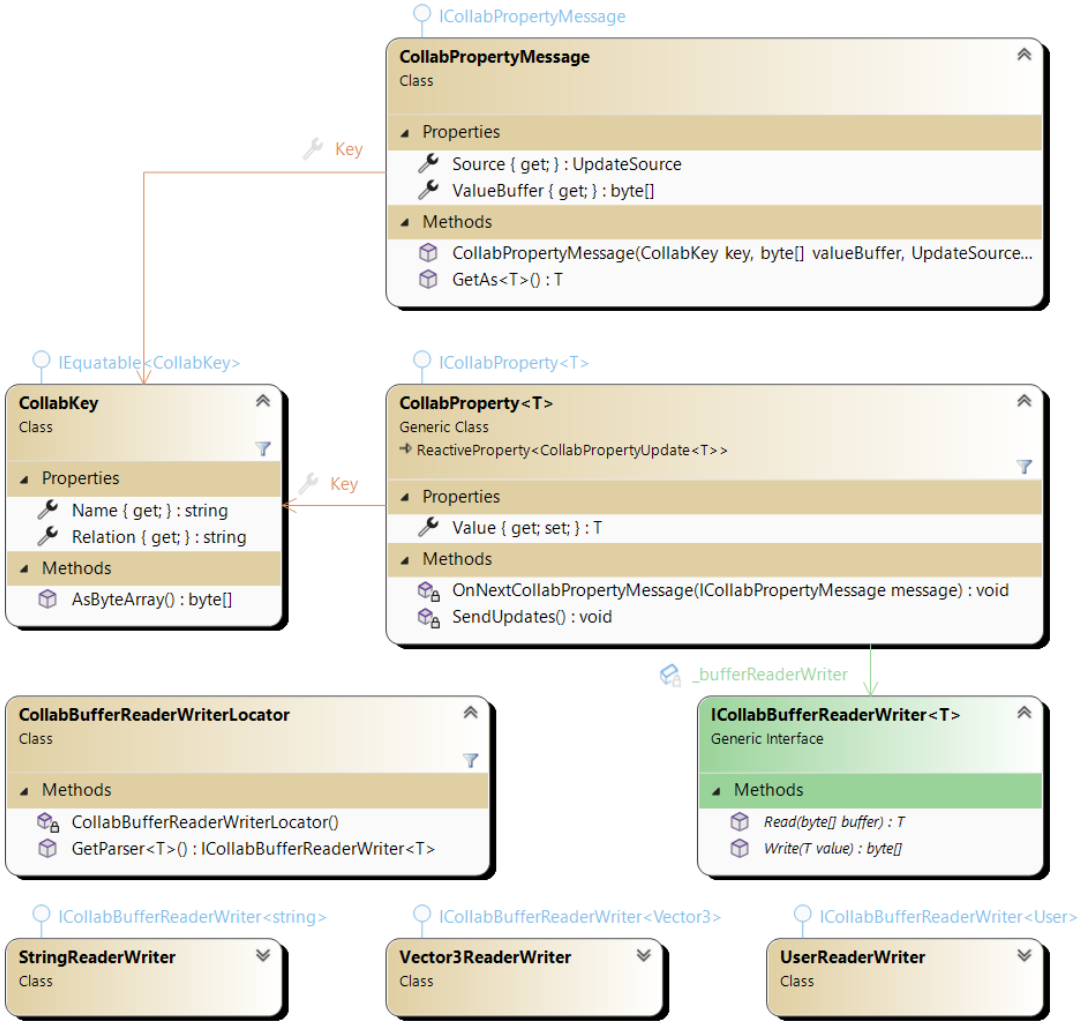

Figure 4.2: Class diagram showing classes and interfaces involved in the serialization of value buffers received and sent to the Collaboration Server. The three implementations of the `ICollabBufferReaderWriter<T>` interface are just examples and various other implementations for other types exist.

4.3.5 Performance Considerations

Handling Incoming Messages

The messages for a specific key-value pair need to be made available to client code in an efficient manner, as there might be hundreds to thousands of key-value pairs. Section 4.3.3 described that incoming messages are grouped by their key using the UniRx GroupBy operator. Internally, the GroupBy operator contains a `Dictionary<string, ISubject<CollabPropertyMessage>>`, which maps each key to an observable that publishes a stream of messages for exactly one key [16]. That means, for each message that is received, the GroupBy operator does a lookup in the map/dictionary of subjects to which it then publishes the message. Since the C# Dictionary is implemented as a hash table, retrieving a Subject/Observable by using its key is very fast, close to $O(1)$ [20]. The exact performance impact of the Dictionary depends on the hashing algorithm being used on the string keys [20]. String hashing algorithms usually need to iterate over the length of the string, making them O(n) in time complexity with respect to the length of the key. Other parts of the application can subscribe to these subjects. The number of subscribers to each subject does not affect the overall performance of receiving messages and routing them to the appropriate observable.

Handling Outgoing Messages

The rate at which messages are sent to the Collaboration Server may be limited arbitrarily. Since the Collaboration Server's *batch* operation type supports sending and receiving multiple *put* and *del* messages/operations at once, it makes sense to buffer messages for at least a short amount of time, before sending them (see Section 4.2.2 for details on *batch* and and other supported operations). This can be achieved easily using the UniRx Buffer operator on the outgoing stream of messages (see Listing 4.1). The Buffer operator transforms an Observable that emits items into an Observable that emits buffered collections of those items [17]. It takes an Observable or TimeSpan as an argument, which indicates how long emitted items of the source Observable should be buffered. The CollabPropertyRepository buffers outgoing messages until every LateUpdate, meaning until all Update functions have run. This makes sure messages are sent with little delay but still allows them to be batched.

```
OutgoingMessages
    .Buffer(Observable.EveryLateUpdate())
    .Where(ops => ops.Count > 0)
    .Select(ops => ops.Select(CreateNetworkBufferFromCollabPropertyMessage))
    .Subscribe(SendMessages);
```

Listing 4.1: Buffer outgoing messages until every LateUpdate and transform them into byte arrays before sending them.

Additionally, the `CollabProperty<T>` class also does not send changes into the outgoing message stream immediately, since when for example the position and rotation of an object is modified every frame, which is the case when it is being moved, this could cause too many messages to be sent. The `CollabProperty<T>` class employs the Sample operator to limit the update transmission rate to a maximum of 20 Hz.

Filtering for Keys

Exact Key In case of Holograms, the exact key for each Hologram property is known on each client, as they are hard-coded into the CollabHologram class. Here, the Observable that provides state updates for a specific `CollabKey` can be obtained by using the `IObservable<ICollabPropertyMessage> GetPropertyObservable(CollabKey key)` method of the `CollabPropertyRepository`, and the value of the message can be parsed using the `ICollabPropertyMessage.GetAs<T>()` method. Alternatively, a `CollabProperty<T>` instance can be used to automatically synchronize a specific key-value pair and also write values to the key-value store.

Multiple Keys For other use-cases, the local client might want to subscribe to multiple key-value pairs on the Collaboration Server at once. For this use-case, the `WatchCollabPropertyMessagesWithFilterTask` and `WatchCollabPropertyMessagesWithRegExTask` allow to watch for messages that match a certain filter function - a Regular Expression in case of the second class. E.g. by using the `WatchCollabPropertyMessagesWithRegExTask` class with the regular expression `"^Hologram\0"`, the corresponding code can subscribe to all key-value pair updates of Holograms. Each key available through the Collaboration Server will only be checked against the filter function once, not for every message received. The `WatchCollabPropertyMessagesWithFilterTask` will filter the Observables that become available and subscribe to the ones that match its filter.

Emitting new Hologram Versions

Every time a Hologram property is updated through the CollabHologram a new Hologram version is emitted. This however also means that while a Hologram is moved, a new Hologram version is registered in the `CollabHologramRepository`. This is a necessity to the approach of providing real-time updates through the existing `IHologramRepository` interface, but may be considered rather inefficient, as new Hologram objects are created for changes in just one property. This happens at rates of up to 20 Hz when Holograms are moved, i.e. their position and rotation properties are updated. It would be possible to mitigate this by rate-limiting newly emitted Hologram versions, but since we want Hologram transforms to update at 10-20 Hz, i.e. at the rates they received updates via the Collaboration Server, additional code would need to be written to update the visual components at those rates.

4.3.6 Collaborative Features

With real-time synchronization of key-value pairs from the Collaboration Server integrated into Hololayer, many collaborative features could now be implemented. For this thesis, the main goal was to update Hologram data in real-time.

Editing and Placing Holograms

Holograms could already be moved and placed through the Hololayer app, but in order to see these changes on other devices all Holograms had to be reloaded through the REST-API. Now, Holograms can be moved through the Hololayer mobile app while all connected clients see these changes in real-time. A newly created Hologram is also loaded on other clients without the need to refresh all Holograms. All Hologram properties can be changed and are transmitted through the collaboration server. One exception persists though, which is Hologram content. The content of a Hologram is a separate arbitrary data buffer, that can contain Hologram-specific data. E.g. an image for a Photo Hologram, or an audio file for an Audio Hologram. Since the Hologram content can be of considerable size, it is not part of the Hologram data structure. While possible, transmitting Hologram content through WebSockets may also quickly cause network congestion. Additionally, the Hololayer client does not support reacting to changes in Hologram content as of yet. Although every Hologram type would need to be able to respond to updated content, Hologram-specific implementations fall outside the scope of this thesis. To support updating Hologram content in real-time, the Hololayer client would need to be refactored, but most of the existing code for loading and parsing the Hologram-specific content could be re-used, just in a different place.

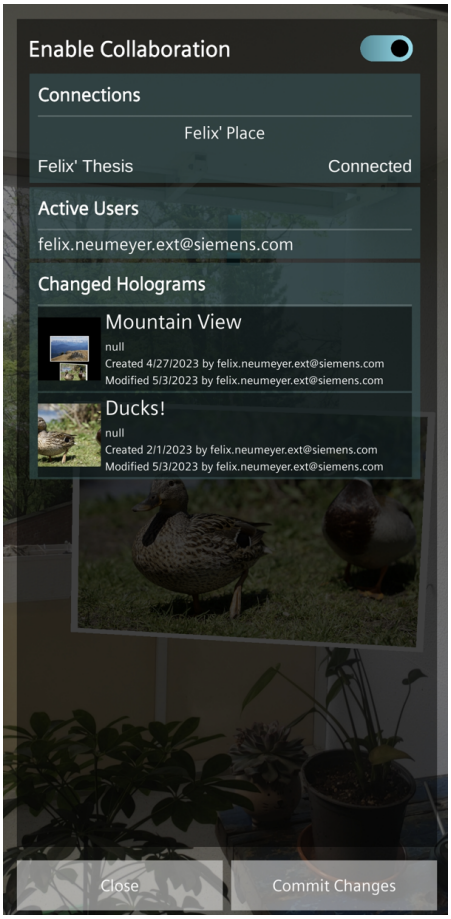

Figure 4.3: Collaboration menu added to the Hololayer iOS and Android clients.

Active Users List

As a demonstration of the client networking solution proposed in this thesis, the `CollabConnectedUserRepository` has been implemented. It keeps track of all currently connected clients on all Collaboration Servers the local Hololayer client is currently connected to. It listens for `CollabPropertyMessages` with `CollabKeys` that match a regular expression by using the `WatchCollabPropertyMessagesWithRegExTask` and publishes the local user on all connected Collaboration Servers by using one `CollabProperty<User>` instance for each server. The list of active users can be found in the Hololayer app's collaboration menu (see Figure 4.3).

Committing Changes

Changes to Holograms are not persisted immediately, when they are made. They are sent to the Collaboration Server, where they are cached. The Holograms that have been modified are listed in the Collaboration menu, allowing all connected users to review them. A commit request can then be sent to the Collaboration Server in order to persist the changes made to modified Holograms through the collaboration menu (see Figure 4.3).

5 Evaluation

5.1 Collaborative Spatial Design User Study

A user study with 12 expert users was conducted following a two-step process, aimed at gathering ideas and opinions on features for a VR-based spatial design application. The study's primary objective was to evaluate the potential usefulness of each tool developed for the VR prototype and to collect user suggestions for additional features. The study's focus was not on UI/UX aspects, but rather on the general potential impact of each feature within a collaborative spatial design context.

The first step of the user study required participants to generate feature ideas independently. This phase was designed to stimulate creative thinking and to avoid biasing participants towards the features already implemented in the prototype. By asking for their input before demonstrating the prototype, we aimed to capture a broad range of ideas based on the participants' experiences and understanding of design processes, which could potentially bring new perspectives and considerations to the development of a VR-based spatial design application.

After participants had brainstormed their own feature ideas, we then transitioned to the second phase of the study. In this phase, participants were shown demonstration videos of the prototype's features. The goal of this demonstration was to give participants a concrete understanding of the prototype's capabilities and to provide a basis for further discussion and feedback.

Participants were asked to evaluate the usefulness of each demonstrated feature and to provide suggestions for improving or expanding the application. This feedback helped us understand not only the perceived usefulness of the features but also potential shortcomings, areas of improvement, and new feature ideas inspired by the prototype. The user study questions can be found in the Appendix.

5.1.1 Participants

The user study was conducted among a diverse group of participants, both in terms of age, gender, and professional background. The participants' ages ranged from 23 to 55 years old, with a median age of 26 years. The gender distribution included 7 males and 5 females, providing a reasonably balanced gender perspective on the

research questions. About half of the participants were employees at Siemens and work on the Hololayer AR system. The other half were either recent graduates, or current students of the Informatics: Games Engineering Master's degree program at the Technical University of Munich.

The occupations of the participants who specified them were diverse, including Unity Developers, an AI Engineer, a Full-Stack Developer, an Engineer, and a PhD Candidate / Digital enterprise expert.

Participants had experience developing on a variety of platforms, including PC, Mobile, Augmented Reality (both head-mounted and hand-held), and Virtual Reality. Some also had experience with console development. Except for one participant, all had experience with either AR or VR.

Most participants were very to extremely familiar with VR technology. The remaining participants indicated a moderate familiarity. The participants' experience with VR applications varied from less than 10 hours to more than 500 hours. Most participants had between 50 and 500 hours of experience.

Out of the 12 participants, 8 had used collaborative or design-related VR applications before, while 4 had not. This suggests a reasonable level of familiarity with the types of applications under consideration in the study.

Their professional or academic backgrounds make them expert users with a high degree of credibility with regards to the questions of this user study.

5.1.2 Use-Case and Feature Brainstorming

The participants came up with a large, diverse number of feature ideas. By grouping similar ideas into categories, we found common feature-sets that seem to be important to collaborative environments. The ideas were not the same for each category, but similar enough to be considered as a common set of functionalities. Since participants rated the importance of their ideas, we can give average importance ratings for each category, however, due to the small sample sizes, they are of little significance.

Real-time Communication and Chat Real-time communication was one of the most frequently mentioned features, receiving the highest average importance **rating of 4.57** by **seven people**. Participants advocated for the inclusion of chat, voice, and emoji functions to facilitate more engaging and dynamic interactions. The integration of both proximity and global chat rooms was also suggested to accommodate different communication needs, including the need to mute other users. Overall I would agree that especially proximity chat would be a very important feature if multiple groups of designers are working in different locations of the virtual environment. It would remove the need to join a separate voice call and seamlessly allow users to join other

groups' conversations by simply teleporting to them. I had included voice chat in my own user stories as well with a maximum importance rating of 5.

Annotations and Feedback With an average importance **rating of 3.71**, the ability to leave notes, comments, and feedback on objects or designs was another desired feature mentioned by **seven people**. Participants saw value in the use of annotations for clarifying design intentions, providing constructive criticism, and facilitating asynchronous communication. Annotations also include 3D drawings, as implemented in the prototype (see Section 3.4.4).

Change Tracking and Version Control The ability to track and visualize user actions over time was a feature suggested by four participants. With an average importance **rating of 3.75**, it was deemed relatively crucial for effective collaboration. This capability would facilitate the review of changes made to a scene or design, contributing to a more transparent and accountable collaboration process. This also includes Undo/Redo capabilities as implemented and described in my prototype (see Section 3.5).

Change Proposals and Approval Slightly different from change tracking, or potentially built on top of it, was a feature suggested by **three participants**. They all mention making suggestions for changes that can be approved by others and give it an average importance **rating of 3.67**. Suggestions may be highlighted visually in the scene. Changes could be viewed, improved and voted on by others.

Access Control and Permissions Access control and permission management was mentioned by **three participants** and received an average importance **rating of 4.33**, highlighting its significance in maintaining the integrity of user contributions. Participants expressed a desire to secure their work from inadvertent or intentional modifications by others, emphasizing the need for robust access control mechanisms in a collaborative environment.

User Location and Movement Knowledge of other users' locations and the ability to move freely within the virtual environment were also deemed important, with an average importance **rating of 3.67**. Features such as minimaps, teleportation to key places or collaborators, and tracking of user movements were suggested to enhance spatial awareness and interaction.

Asset Sharing and Importing The ability to share and import 3D assets was another popular feature set mentioned by **four participants**, with an average importance **rating**

of 4.5. Participants proposed a common asset database for environments and the ability to upload reference images or other resources, indicating a need for versatile asset management capabilities.

Private Workspace or Private Mode Finally, the concept of a private workspace or mode was mentioned, albeit only by **two people**. Receiving an average importance **rating of 3**, this feature would allow users to experiment with designs privately before sharing them with the team. One user said he would for example like to turn lights on/off in the scene without bothering other users.

3D Drawing Participants identified the need for sketching and planning tools, allowing them to draw in 3D, or on a virtual whiteboard.

5.1.3 Virtual Reality Prototype Feature Feedback

After gathering the participants' own ideas, they were shown the features implemented in the prototype as video descriptions. For each feature, they were asked to rate the importance from 1 (Least important) to 5 (Most important), in the same way they rated their own ideas. Since the feature videos show use-cases and explain the reasoning behind the implementation, participants may have rated the prototype features as more important than their own ideas.

Asset Palettes

Study participants rated the of the asset palettes with an average of 4.6. This shows that most participants see this feature as highly important for their work in this environment. The participants generally see the 3D Annotation Pen as a valuable tool for communicating and sharing ideas, creating placeholder drawings, and leaving notes or reminders within the environment. It was also mentioned that this tool can potentially make collaboration more efficient and enhance the level of detail in a design.

The participants generally see the Asset Palette as a crucial feature due to its function as a way to select and place assets, which is a core task in any spatial design application. They also highlight the convenience of the recently used functionality, the intuitive user interface, and the ability to quickly add objects to the scene as reasons for their ratings.

All participants indicate that they would use the Asset Palette feature, and they appreciate its intuitiveness, speed, and how it organizes assets. Some see potential for speeding up their design process and appreciate the ability to manipulate objects' positions and sizes directly in VR.

Suggestions for improving this feature include:

- Implementing a search or filter functionality for easier navigation through assets.
- Enabling physics simulation for asset placement.
- Adding a multi-selection feature to facilitate bulk actions like duplicating multiple objects.
- Adding 3D previews for each asset.
- Providing an option for automatic alignment/placement at the hitpoint of the ray.
- Including a favorites section or bookmarks for assets.
- Incorporating a toggle for the menu's automatic re-opening after asset placement.
- Allowing the option for physics-based placement (optional, to allow for creative placement as well).

Overall, the feedback for the Asset Palette feature is very positive, with participants acknowledging its importance in the spatial design process and providing valuable suggestions for further enhancement.

3D Annotation Pen

The average importance rating for the 3D Annotation Pen feature, based on the scores given by your study participants, is 4.18 (out of 5). This indicates that most participants view this feature as significantly important for their work in the virtual environment.

Participants regard the 3D Annotation Pen as a valuable tool for expressing ideas in a more visual and intuitive manner, especially in a collaborative setting. They find it beneficial for quick prototyping, highlighting parts of the environment, or leaving notes for others. However, some participants noted that while the tool is useful, there might be more efficient alternatives for certain tasks (e.g., using text notes or directly using assets).

Participants expressed enthusiasm about using the 3D Annotation Pen. They appreciate its potential for enhancing communication within the VR space, as well as the creative freedom it offers.

Participants provided several suggestions for enhancing this feature, which include:

- Adding colour options including transparency, brush sizes, and brush types.
- Incorporating an eraser.
- Enabling users to draw certain shapes and straight lines.

- Making drawings temporary or allowing for their bulk deletion.
- Scaling the world or the player for more detailed drawings or quick coverage of large areas.
- Implementing text detection to convert handwritten notes into digital text.
- Adding a ruler tool to draw straight lines
- A toggle button for drawing in fixed depth, so that you can sketch in 2D.

In conclusion, the 3D Annotation Pen is seen as an important and useful tool in the VR environment, with potential improvements suggested for further enhancing its utility and user experience.

Collaborative Undo/Redo

The average importance rating given by the participants for the Collaborative Undo/Redo feature is 4.8 (out of 5). This indicates a general consensus among participants that this feature is fundamental to a smooth and efficient user experience. Participants highlighted the Undo/Redo feature as a necessary component of any software, as it allows for the correction of errors and facilitates experimentation. They particularly emphasized the need for this feature in a collaborative VR setting, where changes might inadvertently affect others' work. Most participants expressed a strong interest in utilizing the Undo/Redo feature, recognizing its importance in reversing mistakes and exploring design ideas. However, they also noted potential complexity in understanding the effect of the "redo" action in a view-based Undo/Redo system.

Participants provided the following suggestions on how to improve this feature:

- Implementing a mode selection for different Undo/Redo methods (as also proposed by me).
- Displaying what will be undone prior to executing the undo action, for better understanding of the implications.
- Showing a preview (ghost object) of the undo position when hovering over the undo button.
- Assigning Undo/Redo permissions based on user roles, with approval required for changes made to others' objects.
- Visualizing active work regions or allowing object selection before offering undo possibilities.

- Providing a list of possible undo items for selection.
- Adding a confirmation or animation for the undo action.

To summarize, the Collaborative Undo/Redo feature is seen as critical by the participants, with potential enhancements suggested to improve user understanding and control over the undo/redo actions. Many of the suggestions are things I have considered as future work myself.

Teleportation

The average importance rating for the Finding Users/Teleportation feature is 4.1 (out of 5). This indicates that most participants view this feature as highly valuable, particularly in scenarios involving large virtual environments or collaboration. Participants deemed the teleportation feature necessary for quickly moving across large virtual spaces, finding collaborators, and maintaining social and professional distances. They noted its importance varies depending on the size and complexity of the scene. While some participants expressed enthusiasm for the teleportation feature due to its potential to save time and facilitate collaboration, others expressed reservations, mainly due to potential disorientation after teleporting.

Participants recommended the following improvements:

- Implementing a teleportation preview or undo feature to prevent accidental teleportation to the wrong person.
- Adjusting the teleportation mechanism to ensure that a teleported user appears within the field of view of the user they teleport to, but not in a blocking position.
- Introducing a map-based solution for visualizing other users' positions relative to one's own and the whole environment.
- Providing a notification to the user being teleported to, to prevent confusion or startlement.
- Implementing a shader to highlight other users through objects.
- Making usernames visible through walls and allowing users to fly through objects.
- Allowing users to choose to teleport above another player, so that they are not startled.

In conclusion, the Finding Users/Teleportation feature is considered important by the majority of participants, particularly for large-scale or collaborative virtual scenarios. Suggestions for improvement focus mainly on enhancing user orientation and awareness during and after teleportation.

Laser Pointer

The average importance rating for the Laser Pointer feature is 4.08 (out of 5). This indicates that participants view this feature as highly beneficial, particularly for communication and measurement in the virtual environment. Participants noted that the Laser Pointer feature is important for pointing and measuring distances in the virtual environment. It serves as an alternative to physical gestures and is particularly useful for highlighting specific locations or objects to other users. Most participants expressed a willingness to use the Laser Pointer feature due to its utility for communication and distance measurement. Some participants also pointed out its usefulness for adhering to spatial constraints in certain tasks.

Participants suggested the following improvements:

- Adding additional functionality such as measuring tape between specified points.
- Allow switching between modes for pointing and measuring.
- Including a toggle to show/hide the distance measurement.
- Enhancing the pointer's visibility with a dot at the end and changing its color upon hitting an object.
- Integrating the Laser Pointer feature with a line drawing tool for temporary scribbles in the air.
- Allowing users to set start and end points for measurements.
- Highlighting the object being pointed at.

In conclusion, the Laser Pointer feature is highly valued by the majority of participants, especially for its dual functionality in pointing and distance measurement. Suggested improvements mainly focus on enhancing visibility, user control, and flexibility in the feature's use.

5.1.4 Using Collaborative VR for Spatial Design Tasks

Regarding the research question whether the participants would want to use a VR-based collaborative spatial design tool like the prototype I developed, **11 out of the 12** participants answered that they would, given further improvements and a more comprehensive feature set. The reasons given varied, but common themes included the potential for improved collaboration, the intuitive nature of VR, and the possibility of more efficient and effective design processes. One participant expressed doubt, preferring a mouse and keyboard for speed, since he is very used to these input

modalities using the Unity Editor. Another participant's response was conditional, suggesting that the usefulness of such a tool depends on the specific project.

One of the main reasons participants want to use such a tool is the potential for **improved collaboration**. Participants believe that a VR-based design tool can facilitate effective communication and teamwork, especially in 3D design projects. The real-time collaboration features can help teams coordinate their work more efficiently, reducing misunderstandings and saving time.

Participants also appreciate the **immersive experience** offered by VR technology. They feel that being able to interact with the 3D design in a virtual space can provide a better understanding of the spatial properties of the design. This is especially important for fields like architecture and interior design, where spatial perception is crucial.

VR technology can provide more **intuitive interaction methods** compared to traditional design software. Participants believe that the ability to manipulate objects in a 3D space in a natural and intuitive way can enhance the design process. This is particularly beneficial for brainstorming and visualizing ideas in the early stages of a design project.

Some participants see the VR-based design tool as a **complementary tool to traditional design processes**. They think that the tool could be used for specific tasks, such as viewing and interacting with the design in a 3D space, while traditional design software could be used for tasks requiring precise control or fast placement of objects.

Participants also mentioned that using a VR-based design tool can be **fun and engaging**. The immersive nature of VR can make the design process more enjoyable, which can boost creativity and motivation. This is especially relevant for game design, where creativity plays a major role.

When analyzing the importance rating of each feature given by the users, there may is a correlation with their hours of VR experience. Eight participants, who have more than 50 hours of VR experience, tend to give higher importance ratings for most of the features. It is particularly noticeable that they consistently rate the Asset Palette, 3D Annotation Pen, Collaborative Undo/redo, and Finding Users/Teleportation features as highly important (4 or 5).

The other four participants, who have less than 50 hours of VR experience, gave mixed ratings. One participant rated the Finding Users/Teleportation feature as very important (4), but gave a lower rating (3) for the Asset Palette. Another participant gave high ratings for the Asset Palette and Finding Users/Teleportation features (both 5), but gave only a 3 for the Laser Pointer. A different participant rated the Finding Users/Teleportation feature as less important (3) while giving high ratings to the other features (all 5). Another participant rated all features as highly important except for the Laser Pointer.

This data suggests that users with more VR experience generally recognize the

importance of these features more than those with less experience. Users with more experience in VR may be a bit biased towards using VR-based tools. It's worth noting that this is a small sample size, so these observations should be considered preliminary. More data would be needed to confirm these trends.

5.1.5 Limitations and Problems

Persistence With the prototype, its currently not possible to persist changes in a useful way. We have tried adding an existing Unity package for saving Play Mode changes, but it did not work with every object or unlinked the placed objects from their corresponding prefabs, making all objects placed or modified within a scene become harder to manage. Since persisting the changes from Unity's Play Mode is another completely separate problem, there was no time to investigate further. For the purpose of the prototype - finding and testing important features for collaborative spatial design - it was not necessary to fix it, but for a real application it is indispensable.

User Study Participants While we had a diverse set of participants ranging from very experienced full-time employees at Siemens to Games Engineering students in pursuit of their Master's degree, with a sample size of only 12, the generalizability of the study's findings is limited. All participants either had a lot of experience with VR or with computer games. The user study was designed to target experienced or expert users with regards to VR, in order for them to be able to comment on the importance of each feature and the usability of collaborative VR in spatial design tasks, but that also means that they may be biased towards using a VR-based tool. Additionally, all study participants know me. Some are colleagues, others are friends. Therefore their feedback could be more positive than that of strangers.

Prototype Videos The videos demonstrating the prototype features explained every feature in detail with use-cases and additional information about usability considerations. User study participants may have rated the features of the prototype more favorably since they were presented in a positive manner. If users had actually tried the prototype themselves without detailed explanations, the feedback may have been more mixed. Still, since we did not want to evaluate the UX or UI of the prototype, showing demonstration videos seems like an effective method of collecting user feedback and ideas for improvement.

5.2 Hololayer Collaboration

5.2.1 End-to-end latency

We measured the end-to-end latency between two Hololayer iOS clients, from a modification being made on one client, to the modification becoming visible on the other. The end-to-end latency therefore includes processing time on the modifying client until a message is sent over WebSockets to the Collaboration Server, the processing time on the Collaboration Server until the state is cached in the key-value stored, the message being sent to other clients, the processing time on the receiving client and finally rendering the change in Unity's game loop. We took screen-recording of the Hololayer iOS client, while modifying the rotation of a Hologram. Within the view of the recording client, we held a second phone, which observed the Hologram being modified (see Figure 5.1). By analyzing the screen-recording frame-by-frame, we calculate the latency between the Hologram being modified on the recording client and the second client showing the result of the modification.

The delay between the client doing to the modification and the other client displaying the modification was 7-8 frames at a frame-rate of 33.12. The end-to-end latency is therefore between 211 to 240 milliseconds, with the Collaboration Server deployed on AWS with a ping round trip time of just 12 milliseconds. Note that messages are sent with delays of up to 50 milliseconds because each `CollabProperty<T>` uses a `UniRx.Sample` operator at 20 Hz to limit the number of outgoing messages per property. This could certainly be implemented differently, allowing each property update to be sent immediately if there had not been an update within the last 50 milliseconds.

5.2.2 Limitations and Problems

Key Size

Since the Collaboration Server is a key-value store and we need to send the key with every *put* or *delete* or message, the key takes up most of the data within a message, as we synchronize single Hologram properties, which are usually comprised of rather simple and small data types. For example, two of the properties that are being modified most frequently are the *position* and *rotation* properties, which are both represented by three float values each. Therefore, the values of each property take up $3 * 4 = 12$ bytes, but the keys are represented as a strings. In both cases, the key is 55 bytes long (see Section 4.2.3. The key size could be reduced in various ways.

GUID Representation The Hologram's Guid is serialized as a string within the key, even though a GUID is actually defined as a 16 bytes integer [21]. The human-readable

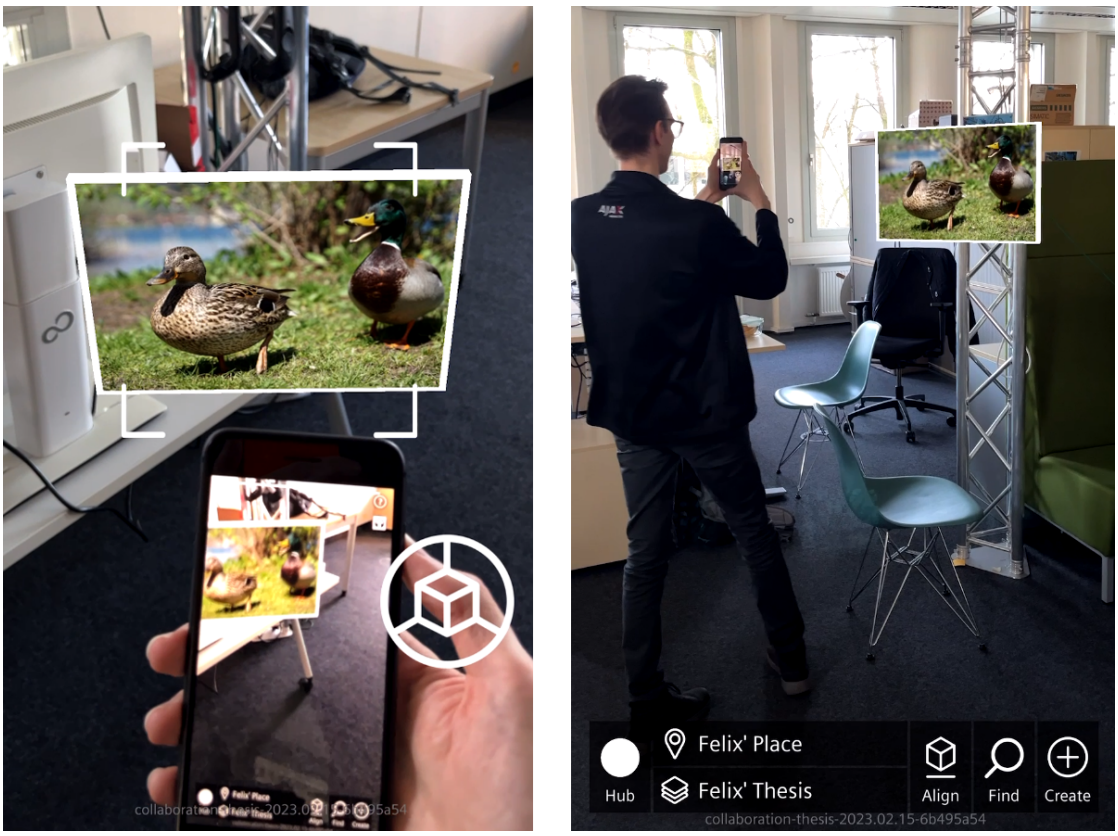

Figure 5.1: Hololayer client running on two iPhones, where one person is moving an Image Hologram.

string representation of the GUID is a series of lowercase hexadecimal digits in groups of 8, 4, 4, 4, and 12 digits and separated by hyphens [21]. It's therefore taking up $8 + 4 + 4 + 4 + 12 + 4 * 2 = 36$ bytes as a UTF-8 string within the key, instead of just 16 bytes in its integer representation. However, even when replacing the GUID representation, the key's relation and property name parts being strings would still make the key portion of each message relatively large, compared to the value. For a Hologram position property, the reduced size of the key would be 35 bytes instead of 55 bytes, which is still almost three times the size of the `Vector3` value size at 12 bytes. The total message size for one position property update would be decreased from currently 76 bytes to 56 bytes, which is a **reduction of 26.3%**.

Key Aliases A very efficient way of transmitting key references is adding support for key aliases, which were also introduced in version 5 of the MQTT protocol as Topic

Aliases [22]. A topic alias in MQTT5 "is an integer value that is used to identify the Topic instead of using the Topic Name" [22]. The MQTT5 specification states that "this reduces the size of the PUBLISH packet, and is useful when the Topic Names are long and the same Topic Names are used repetitively within a Network Connection", which is exactly the case with properties like Hologram position or rotation, which are often transmitted up to 20 times per second [22] for a single Hologram. The Collaboration Server could publish a mapping from a *long* to the actual key. E.g. the alias could be defined as a key-value pair in the relation "Alias", followed by the target key:

```
Alias{0x00}Holograms{0x00}{HologramGUID}{0x00}position = 1
```

The alias could then be used in place of the actual key by both client and server, reducing the message size drastically. Since the first byte of a Collaboration Server message contains the message type, for each existing message type, a corresponding alias message type could be defined. For a Hologram's position or rotation attribute, the message size for a single (alias) put message could be decreased from currently 76 bytes to just 21 bytes, which is a significant **reduction of 72.4%**. Of course, the key aliases would need to be transmitted in advance, also requiring bandwidth, but only once. For property keys that may be transmitted hundreds of times, that is reasonable. The client application may specify for which properties aliases should be created.

6 Conclusion

6.1 Spatial Design Prototype

We have demonstrated how a VR-based spatial design application could be developed and used, mainly targeting level design for games. The user study has shown that people with more than 50 hours of experience in VR would enjoy using VR-based spatial design tools like the prototype developed in this thesis in addition to their usual workflows. A VR-based editor may not be able to completely replace established 2D Editor tools based on keyboard and mouse input, but can supplement them well. Most user study participants rate the features and tools developed in the prototype as highly important.

We have found comprehensive groups of features that are essential for collaborative VR environments, like real-time communication and chat, tools for effective communication like laser pointers, annotations and feedback (text, audio, image, drawings), change tracking and version control including change proposals and approval, access control and permissions, travel and orientation within the virtual environment including teleportation to other collaborators, asset sharing and importing to add new assets to the pre-defined assets and private workspaces separate from the shared environment. By combining the prototype features with the feedback and ideas of the user study participants, a proper collaborative spatial design application could be developed.

Undo/Redo is one of the most common functionalities in any editing software, but complex to implement in multi-user environments. The Undo/Redo methods implemented or proposed, especially the View-based method, enable intuitive collaboration in 3D environments. They solve some of the problems inherent with reversible changes in multi-user 3D environments, like being able to revert other users' changes while not interrupting their workflow.

The participants of the user study praised the usability considerations of some of the features, like "recent assets" or "repeated placement" in the Asset Menu, or spawning objects far enough away to not feel uncomfortable. All features of the prototype were received very well by the user study participants, though there were many suggestions for improvements.

6.2 Hololayer Collaboration

In this work, I have introduced real-time collaborative features to an industrial Augmented Reality (AR) system known as Hololayer. Originally, the AR client interacted with the backend server through a REST API, where requests were made on demand. This meant that any state changes applied to the underlying data, such as Hologram data, were not observed until a user triggered another update. By integrating this approach with a real-time distributed key-value store, namely the Collaboration Server, I have successfully implemented live updates of Holograms by implementing reactive design patterns. Instead of retrieving all Hologram state through the REST API, partial state updates to specific Hologram properties are shared and applied on all clients connected to the Collaboration Server.

The architecture developed for enabling real-time collaboration within the Hololayer client is well-structured, organized, and purpose-driven. By leveraging reactive programming with UniRx, the implementation efficiently handles asynchronous data streams, resulting in a system that can respond to real-time updates of Holograms. Place, Layer, and other data could be synchronized through the Collaboration Server in a similar fashion.

The modular design of the components facilitates a clear separation of concerns. The `CollabPropertyRepository` manages connections and message handling, while the `CollabProperty<T>` and `CollabHologram` components focus on synchronizing individual properties of Holograms. The `CollabHologramRepository` oversees the collection of Holograms and allows them to be created, fetched, updated or deleted from within other components of the Hololayer client.

The utilization of existing interfaces, such as the `IHologramRepository`, enhances the system's flexibility and extensibility. This design choice allows for easier integration of new features or modifications to existing ones, without causing significant disruptions to the overall codebase. Implementing the `CollabHologramRepository` required minimal intervention in the existing codebase.

Overall, the proposed architecture serves as a foundation for the Hololayer client, effectively addressing the need for real-time collaboration in industrial applications. The implementation enables the development of more complex collaborative features, like the ones discussed for the Spatial Design Prototype. As with any system, there is room for further improvement and expansion to cater to evolving requirements.

6.3 Future Work

6.3.1 Collaborative Features

Given the large amount of features both the user study participants and myself came up with, there are many things that could be implemented to create a better, more complete version of this prototype or an actual product. Depending on the use case, different features should be prioritized. Before deciding on which features are the most important ones, a more in-depth user study could be done with the corresponding target audience, asking participants which features from a list of features they find most important. However I would argue that all of the features proposed in my thesis are important to a collaborative spatial design environment. Here are a few things I would have liked to implement:

- Proximity voice chat with the possibility to also speak to all connected users.
- Collaborative menus, i.e. menus that can be controlled by multiple users. For example the Asset Palette could potentially be scrolled through by multiple users.
- Adding undo/redo previews to my implementation. Each undo/redo command could have its own implementation on how to visualize its actions.
- Support for making change proposals that can be approved by other collaborators. Each proposal could have multiple variations that collaborators could switch between and vote on.

6.3.2 Additional User Studies

We have been able to collect a large number of important features that would make a collaborative spatial design application useful. However we have yet to evaluate the proposed features in comparison with traditional editor tools. For example, a user study could be designed to measure efficiency at certain level design tasks, like furnishing an appartment, comparing users working within the Unity Editor and users in the VR prototype.

6.3.3 Hololayer Collaboration

Non-stateful and Transient Data Messages While the current collaboration architecture relies on key-value pairs to be stored on the Collaboration Server for synchronizing the state of Holograms, there are real-time collaboration features that may necessitate the implementation of additional functionality. For example, certain data types or

actions may not require persistent storage on the Collaboration Server, but must simply be forwarded to other clients in real-time. In such cases, implementing a mechanism for transmitting transient data or events, which are not stored on the server, can be beneficial. This could involve introducing a dedicated communication channel or protocol for sharing temporary or real-time information, such as user interactions or live audio streams among clients.

Support for Layers and Places In this thesis, we only enabled real-time collaboration on Holograms. Places and Layers are still only updated on Hololayer clients when they actively trigger an update of their data repositories, e.g. by opening the list of Layers or Places. Supporting Places and Layers could be achieved in the same way that Holograms are synchronized between clients.

Blocking Simultaneous Key-Value Writes Currently, all clients can write to a key whenever they want, which may happen to problems when two clients try to modify a Hologram simultaneously. The Collaboration Server should provide a locking or ownership functionality, allowing clients to request temporary exclusive write access to a specific key or range of keys. The Unity Netcode for GameObjects library requires clients to request Ownership of NetworkObjects before they can modify the object's properties.

List of Figures

List of Tables

Bibliography

[1] Y. Huang, S. Shakya, and T. Odeleye. "Comparing the functionality between virtual reality and mixed reality for architecture and construction uses". In: *Journal of Civil Engineering and Architecture* 13.1 (2019), pp. 409–414.

[2] V. Lehrbaum, A. MacWilliams, J. Newman, N. Sudharsan, S. Bien, K. Karas, C. Eghtebas, S. Weber, and G. Klinker. "Enabling Customizable Workflows for Industrial AR Applications". In: *2022 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2022, pp. 622–630. DOI: 10.1109/ISMAR55827.2022.00079.

[3] U. Exner and D. Pressel. *Basics Spatial Design*. Berlin, Boston: Birkhäuser, 2017. ISBN: 9783035612394. DOI: doi:10.1515/9783035612394. URL: https://doi.org/10.1515/9783035612394.

[4] NVIDIA. *Holodeck Product Website*. 2017. URL: https://www.nvidia.com/en-us/design-visualization/technologies/holodeck/.

[5] Unity. *EditorXR Website*. URL: https://unity.com/de/editorxr.

[6] L. Beever, S. Pop, and N. W. John. "LevelEd VR: A virtual reality level editor and workflow for virtual reality level design". In: *2020 IEEE Conference on Games (CoG)*. 2020, pp. 136–143. DOI: 10.1109/CoG47356.2020.9231769.

[7] J. Choi. "Merging Three Spaces: Exploring User Interface Framework for Spatial Design in Virtual Reality". MA thesis. 2016.

[8] C. Rendl. "Regionales Undo/Redo für Multi-User-Anwendungen auf großen interaktiven Oberflächen [Regional Undo/Redo for Multi-User-Applications on Large Interactive Surfaces]". MA thesis. 2011.

[9] S. Weber, M. Ludwig, and G. Klinker. "Ubi-Interact: A modular approach to connecting systems". In: *EAI Endorsed Transactions on Mobile Communications and Applications* 6.19 (July 2021). DOI: 10.4108/eai.14-7-2021.170291.

[10] Unity. *Netcode for GameObjects*. URL: https://docs-multiplayer.unity3d.com/.

[11] Unity. *XR Interaction Toolkit Manual*. URL: https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.3/manual/index.html.

[12] R. Nystrom. *Game Programming Patterns, Command Pattern*. 2014. URL: http://gameprogrammingpatterns.com/command.html.

[13] HiveMQ. *MQTT Publish/Subscribe Pattern*. 2015. URL: https://www.hivemq.com/blog/mqtt-essentials-part2-publish-subscribe/.

[14] N. Pereira, A. Rowe, M. W. Farb, I. Liang, E. Lu, and E. Riebling. "ARENA: The Augmented Reality Edge Networking Architecture". In: *2021 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. 2021, pp. 479–488. DOI: 10.1109/ISMAR52148.2021.00065.

[15] R. C. Martin and J. O. Coplien. *Clean code: a handbook of agile software craftsmanship*. Upper Saddle River, NJ [etc.]: Prentice Hall, 2009. ISBN: 9780132350884 0132350882. URL: https://www.amazon.de/gp/product/0132350882/ref=oh_details_o00_s00_i00.

[16] Y. Kawai. *UniRx (Reactive Extensions for Unity)*. URL: https://github.com/neuecc/UniRx.

[17] .NET Foundation. *Reactive Extensions (Rx)*. URL: https://github.com/dotnet/reactive.

[18] .NET Foundation. *ReactiveX Online Documentation*. URL: https://reactivex.io/documentation/operators.html.

[19] Google. *Protocol Buffers*. 2008. URL: https://protobuf.dev/.

[20] Microsoft. *.NET Standard 2.1 - System.Collections.Generic.Dictionary - Online Documentation*. URL: https://learn.microsoft.com/en-us/dotnet/api/system.collections.generic.dictionary-2?redirectedfrom=MSDN&view=netstandard-2.1.

[21] Microsoft. *.NET System.Guid Documentation*. URL: https://learn.microsoft.com/de-de/dotnet/api/system.guid?view=netstandard-2.1.

[22] OASIS Open. *MQTT 5 Specification*. 2019. URL: https://docs.oasis-open.org/mqtt/mqtt/v5.0/os/mqtt-v5.0-os.html.

Appendix

6.1 Collaborative Spatial Design User Study

Collaborative Spatial Design in Virtual Reality (VR)

In this user study, we are collecting ideas and opinions on features for a VR-based spatial design application. Spatial design encompasses a variety of fields, such as level design in game development, interior or architectural design, industrial planning, and other disciplines that involve designing 3D environments. As part of my master's thesis, I have developed a prototype using the Unity game engine that showcases several features and tools specifically designed for collaborative spatial design processes.

The primary objective of this user study is to evaluate the usefulness of each tool in a collaborative spatial design application and gather user suggestions for additional features. Please note that we are not focusing on the user interface (UI) or user experience (UX) aspects of the application, but rather on the general usefulness and potential impact of each feature within a collaborative spatial design context.

First, we will ask you to come up with feature ideas yourself. Afterwards, you will be shown demonstrations of the prototype's features, and we will ask for your feedback on their usefulness and any suggestions you may have for improving or expanding the application. Your input is invaluable in helping us refine the prototype and better understand the needs of users in the spatial design field.

* Gibt eine erforderliche Frage an

1. Informed Consent Form *

Wählen Sie alle zutreffenden Antworten aus.

- [ ] I have had the opportunity to ask any questions related to this study, and received satisfactory answers to my questions, and any additional details I wanted.
- [ ] I have been informed that I can abort the questionaire any time without needing to provide reasons.
- [ ] I understand that the collected data taken during this questionaire or subsequent interviews will be evaluated for this research.
- [ ] I understand that relevant sections of the data collected during the study may be looked at by individuals from the research group Augmented Reality of TU Munich and/or Siemens, where it is relevant to my taking part in this research. I give permission for these individuals to have access to my responses.
- [ ] I am aware that excerpts from this questionaire or subsequent interviews may be included in publications to come from this research. I grant permission to use direct quotes of my answers. Quotations will be kept anonymous.

Demographic Questions

2. How old are you? *

_____

3. Please select your gender. *

Markieren Sie nur ein Oval.

- ◯ Female
- ◯ Male
- ◯ Non-binary
- ◯ Prefer not to answer
- ◯ Sonstiges: _____

4. What is your current employment status? *

Markieren Sie nur ein Oval.

- ◯ Student (not employed)
- ◯ Working student (Werkstudent)
- ◯ Full-Time Employee
- ◯ Part-Time Employee
- ◯ Self-employed
- ◯ Unemployed
- ◯ Prefer not to answer
- ◯ Sonstiges: _____

5. Optionally specify your occupation (E.g. Software Engineer, Unity Developer, ...)

_____

6. Are you a left- or right-handed? *

Markieren Sie nur ein Oval.

- ◯ Left-handed
- ◯ Right-handed
- ◯ Both

7. How frequently do you play video games on average per week? *

Markieren Sie nur ein Oval.

- ◯ Not at all
- ◯ < 5 hours
- ◯ 5-10 hours
- ◯ 10-15 hours
- ◯ 15-20 hours
- ◯ > 20 hours

8. For which platforms have you developed software or games in the past, including non-programming tasks like UX design? *

Wählen Sie alle zutreffenden Antworten aus.

- ☐ Augmented Reality (Head-mounted)
- ☐ Augmented Reality (Hand-held)
- ☐ Virtual Reality
- ☐ PC
- ☐ Consoles
- ☐ Mobile
- ☐ Sonstiges: _____

9. How familiar are you with Virtual Reality (VR) technology? *

Markieren Sie nur ein Oval.

- ◯ Not familiar at all
- ◯ Slightly familiar
- ◯ Moderately familiar
- ◯ Very familiar
- ◯ Extremely familiar

10. How many hours of experience do you have using VR applications? (approximate) *

Wählen Sie alle zutreffenden Antworten aus.

- ☐ Less than 10 hours
- ☐ 10-50 hours
- ☐ 50-100 hours
- ☐ 100-500 hours
- ☐ More than 500 hours
- ☐ Sonstiges: _____

11. Have you used any collaborative or design-related VR applications before? (e.g., social VR platforms, Gravity Sketch, Tilt Brush, etc.) *

Markieren Sie nur ein Oval.

- ◯ Yes
- ◯ No

Brainstorming: Use Cases and Tools for Collaborative Spatial Design Tasks in XR

In this part of the study, we ask you to come up with key use cases and tools that you consider most beneficial for collaborative spatial design tasks in Virtual or Augmented Reality, like level design for games, interior design, architecture, industrial planning or other spatial design fields. Imagine you want to design a room or an open space in virtual reality, or you want to annotate a real-world environment using augmented reality. Maybe also think about collaborative tools or software you have used outside of spatial design tasks and how their features could translate into 3D/XR.

The images below illustrate some environments you could be performing spatial design tasks in, but you can of course think of something else.

Fill out as many or few of the boxes below as you like - they are just placeholders to provide enough space.

Sample Idea:
I'm using a collaborative spatial design application in VR and want to find other users in the 3D scene. This can be achieved by teleporting to them.

(Non-exhaustive) Sample Scenarios

12. Idea #1

13. Idea #1 Importance

Markieren Sie nur ein Oval.

	Least Important				Most Important
	1	2	3	4	5

14. Idea #2

31. Idea #10 Importance

Markieren Sie nur ein Oval.

Least Important

1 ⬭

2 ⬭

3 ⬭

4 ⬭

5 ⬭

Most Important

Prototype Feedback

In the following sections, you'll be shown descriptions and videos of features of a collaborative level design prototype for the Unity Game Engine, which was developed for this thesis. The goal is not to evaluate the user interface (UI) or user experience (UX) of the prototype, but the general usefulness/importance of each feature and how they could be improved. If some UI/UX strikes you as especially good or bad, or is inherent to the feature described, go ahead and comment on it if you want to. Please watch the videos to fully understand the features. The textual feature summaries below the videos are optional to read - they may not cover all aspects of the video but can provide additional information.

Asset Palettes (Prefabs)

Demonstration Video

http://youtube.com/watch?v=50RvQ-fz97U

Feature Summary

The Asset Palettes are pre-defined assets that can be spawned by every collaborator in a networking session. These assets are organized into categories based on the folders they are located in, where the folder name is considered the category, so that they can be added to the Asset Menu of the prototype quickly. Every asset needs to have a NetworkObject and CollabObject component in order to be included in the asset palettes, but adding these components to the asset can be done automatically inside the Unity Editor through Editor Scripts that will prepare every prefab in a certain path. The Asset Menu has a "Recent" tab that contains the recently used assets. It does not only contain the assets you previously used, but also assets used by other users in the current collaboration session. This allows you to quickly re-use assets you had picked out before, or find the ones being used by your collaborators. By enabling "Repeated Placing" the same asset can be placed again without having to re-select this asset from the asset palette menu.

32. Importance Rating *

Markieren Sie nur ein Oval.

Least Important

1 ⬭

2 ⬭

3 ⬭

4 ⬭

5 ⬭

Most Important

33. Please explain your reasoning behind the rating you gave for the importance of this feature. *

34. Would you want to use this feature? Please elaborate. *

35. How would you improve this feature? Would you implement it differently or replace it with something else? *

3D Annotation Pen

Demonstration Video

http://youtube.com/watch?v=ijuU_dHFAPE

Feature Summary

Users can draw 3D lines using the pen annotation tool. They can add text or any other shape they can imagine. For example, they may draw temporary assets for which there is no real asset yet, or they can annotate existing assets that need changes. By drawing collaboratively, multiple users can iterate on design ideas.

36. Importance Rating *

Markieren Sie nur ein Oval.

Least Important

1

2

3

4

5

Most Important

37. Please explain your reasoning behind the rating you gave for the importance of this feature. *

38. Would you want to use this feature? Please elaborate. *

39. How would you improve this feature? Would you implement it differently or replace it with something else? *

Collaborative Undo/Redo

Demonstration Video

http://youtube.com/watch?v=3779hus1BFA

Feature Summary

Reverting or re-applying changes can be tricky when multiple users are working in the same environment. The prototype has two different algorithms to specify which objects to target for Undo/Redo. The **Local Undo/Redo** algorithms allows the user to undo/redo only his own changes. The **View-based Undo/Redo** algorithm allows to revert or re-apply modifications that are currently within the view of the user, independently of who authored the modification. The next Undo/Redo target is highlighted when the user hovers the Undo/Redo button with a controller's ray interactor.

40. Importance Rating *

Markieren Sie nur ein Oval.

Least Important

1

2

3

4

5

Most Important

41. Please explain your reasoning behind the rating you gave for the importance of this feature. *

42. Would you want to use this feature? Please elaborate. *

43. How would you improve this feature? Would you implement it differently or replace it with something else? *

Finding Users (Teleporting)

Demonstration Video

http://youtube.com/watch?v=eZot_G6sSEw

Feature Summary

In order to quickly find other collaborators in the virtual 3D scene, there's a user list in the main menu, through which you can teleport to a user. The teleportation algorithm tries to place you in a way that you do not obstruct the view of the user you're teleporting to by prioritizing locations to the left/right with respect to his viewing direction of the target user. If possible, it also tries to place you at a socially comfortable distance away from the target user, starting with a radius of 1.8 meters and only placing you closer, if there's not enough space. The algorithm also expects a minimum distance to other objects of 0.5 meters, so that you're not placed halfway into a wall.

44. Importance Rating *

Markieren Sie nur ein Oval.

Least Important

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Most Important

45. Please explain your reasoning behind the rating you gave for the importance of this feature. *

46. Would you want to use this feature? Please elaborate. *

47. How would you improve this feature? Would you implement it differently or replace it with something else? *

Laser Pointer

Demonstration Video

http://youtube.com/watch?v=LtLN8MxThgY

Feature Summary

The Laser Pointer feature allows users to point in any direction and at objects with their controller. A red line is rendered from the controller forward. It can be used to coordinate with other collaborators and to measure the distance from the controller to the object hit with the laser. Measured values are visible to other users, too.

48. Importance Rating *

Markieren Sie nur ein Oval.

Least Important

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Most Important

49. Please explain your reasoning behind the rating you gave for the importance of this feature. *

50. Would you want to use this feature? Please elaborate. *

51. How would you improve this feature? Would you implement it differently or replace it with something else? *

Final Thoughts

52. Now that you have seen the prototype's features and came up with your own ideas, what do you think are the most important features and use-cases, including your own ideas? *

53. Considering your current or potential involvement in spatial design projects, would you want to use a VR-based collaborative spatial design tool like the one presented in this study, assuming further improvements and a more comprehensive feature-set? Please explain your reasoning. *