Research Group Database and Information Systems

Development Framework for Cross-Platform Multi-User AR Applications

Master's Thesis Proposal

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by
Christian Fischer

Supervisors:
Dr. Enes Yigitbas
Ivan Jovanovikj

Examiners:
Dr. Enes Yigitbas
Prof. Dr. Gregor Engels
Paderborn, December 21, 2020

UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Eidesstattliche Versicherung

Nachname Fischer Vorname Christian

Matrikelnr. 6512858 Studiengang Wirtschaftsinformatik Master

○ Bachelorarbeit ◉ Masterarbeit

Titel der Arbeit Development Framework for Cross-Platform Multi-User AR Applications

☐ Die elektronische Fassung ist der Abschlussarbeit beigefügt.

☒ Die elektronische Fassung sende ich an die/den erste/n Prüfenden bzw. habe ich an die/den erste/n Prüfenden gesendet.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit (Ausarbeitung inkl. Tabellen, Zeichnungen, etc.) selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Abschlussarbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen. Die elektronische Fassung entspricht der gedruckten und gebundenen Fassung.

Belehrung

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist die Vizepräsidentin / der Vizepräsident für Wirtschafts- und Personalverwaltung der Universität Paderborn. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz NRW in der aktuellen Fassung).

Die Universität Paderborn wird ggf. eine elektronische Überprüfung der Abschlussarbeit durchführen, um eine Täuschung festzustellen.

Ich habe die oben genannten Belehrungen gelesen und verstanden und bestätige dieses mit meiner Unterschrift.

Ort Paderborn Datum 21.12.2020

Unterschrift C. Fischer

Datenschutzhinweis:

*Die o.g. Daten werden aufgrund der geltenden Prüfungsordnung (Paragraph zur Abschlussarbeit) i.V.m. § 63 Abs. 5 Hochschulgesetz NRW erhoben. Auf Grundlage der übermittelten Daten (Name, Vorname, Matrikelnummer, Studiengang, Art und Thema der Abschlussarbeit) wird bei Plagiaten bzw. Täuschung der*die Prüfende und der Prüfungsausschuss Ihres Studienganges über Konsequenzen gemäß Prüfungsordnung i.V.m. Hochschulgesetz NRW entscheiden. Die Daten werden nach Abschluss des Prüfungsverfahrens gelöscht. Eine Weiterleitung der Daten kann an die*den Prüfende*n und den Prüfungsausschuss erfolgen. Falls der Prüfungsausschuss entscheidet, eine Geldbuße zu verhängen, werden die Daten an die Vizepräsidentin für Wirtschafts- und Personalverwaltung weitergeleitet. Verantwortlich für die Verarbeitung im regulären Verfahren ist der Prüfungsausschuss Ihres Studiengangs der Universität Paderborn, für die Verfolgung und Ahndung der Geldbuße ist die Vizepräsidentin für Wirtschafts- und Personalverwaltung.*

Abstract.
Co-located Augmented Reality integrating multiple end devices is recently provided by several applications, for example Google's mobile app *Just a Line*. The application lets users draw simple three-dimensional lines inside the AR scene, visualized consistently on all participating devices. Consistent AR views enhance the users' ability to collaborate. However, such applications are still rare and a framework is missing, which provides features necessary to enable co-located, markerless multi-user AR with consistent AR scenes for domain-specific applications. Implementing those features comes with complications and high effort of time. In this thesis, we present *XshARe*, a framework supporting interactive multi-user cross-platform AR applications. The framework provides synchronization of AR scenes and scene-related meta data. Furthermore, generic interaction modalities for handheld devices are implemented, extendable for HMD support. To enable cross-device communication, a network architecture is given to manage consistent model data and business logic calculation. A current state-of-the-art method to enable AR scene synchronization across multiple devices is the concept of anchor sharing, which is implemented and supported by an extern cloud service. An anchor is a fixed point detected in the real world. Sharing anchors as reference points enables the synchronization of coordinates across devices with differing coordinate systems for the common AR scene. To validate the solution, two example applications using the XshARe framework were developed as case studies.

Contents

1 Introduction

This chapter introduces the reader to the thesis and gives an overview over the thesis' main topic. First, the work's motivation is described to make clear, why there is a need to provide the thesis' project. Afterwards, the thesis' problem is described in detail, including challenges to be overcome. This is followed by a raw description of the developed solution. The chapter closes with an overview over the structure of the following parts.

1.1 Motivation

In recent years, Augmented Reality has made huge steps in technical progress and begun to gain more and more popularity. Augmented Reality (AR) describes computer-based approaches to enrich real environments with virtual elements. Azuma refines this definition by restricting it on cases "in which 3-D virtual objects are integrated into a 3-D real environment in real time" [Azu97]. To create a convincing illusion, digital objects have to remain at the same respective angle, size, and position in the room, even if the user's view is moving. Therefore, the AR device has to scan the area of interest to derive a 3D map from it and later localize coordinates of virtual objects as well as of the AR device's own position. Typical areas of application for AR technology mentioned by Azuma are "medical, manufacturing, visualization, path planning, entertainment and military" [Azu97]. As those can be fields of social interaction, there is the demand to make collaborative, multi-user AR applications possible. That means, virtual objects have to be consistent and synchronous in the view of every participating device. Current AR Software Development Kits (SDKs) provide such features. However, AR frameworks are mostly single-platform oriented and limited on collaborative features concerning visuals, not meta data or business logic. Companies like Apple are interested in selling their hardware devices and therefore making developed software only applicable on their own devices as unique selling points. This makes cross-platform support difficult. Some available approaches, though, offer cross-platform AR experience, but integrated in their own specific application, not in an open, generically usable framework.

As an example, consider an entertainment app for playing the game Mill (or 9 men's morris), as illustrated in Figure 1.1. It consists of a board and up to 18 game pieces. As board games are naturally played on tables, it is intuitive to orientate the board to a table, that means to a horizontal surface above ground. As an alternative, the board could also be visualized floating above ground vertically or horizontally or attached to a wall, that means to a vertical surface.

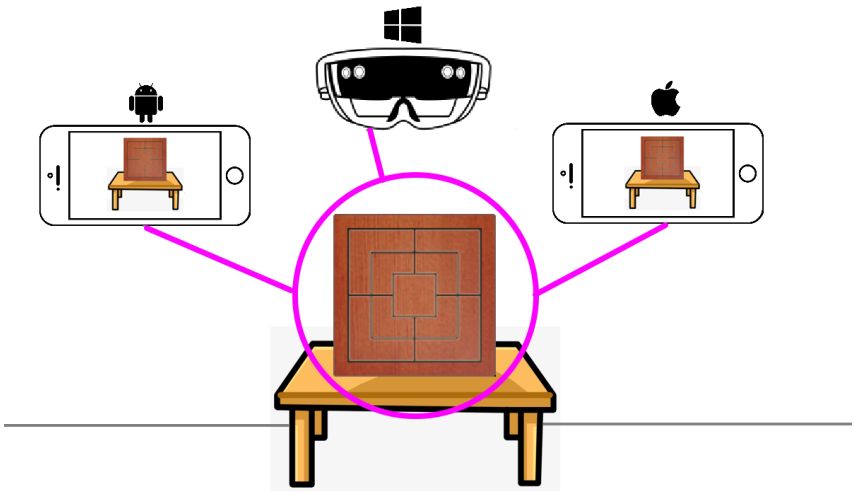

Figure 1.1: Multi-user cross-platform Augmented Reality: Three devices view the same virtual object.

A session of mill requires two participants as players, that means users with the permission to manipulate data within the game scene. Further participants as observers without permission to manipulate are conceivable. All of those users require their own AR device to view the AR scene and possibly interact with it. Players need to be able to place new pieces on the board, move them and remove them, if the game rules allow it. To manipulate a specific game piece and not the whole scene at once, virtual elements have to be selectable and de-selectable. Mandatory for an immersive AR experience is the board being visualized at the same position in real world for each user. This way, participants can collaborate more easily. For example, they can discuss about upcoming moves by pointing with their hands on virtual game pieces and positions. To enable such a synchronization of AR scenes for multiple users, there are several basic steps to run through: First, all participants in the room have to establish a connection to each other. After that, they have to synchronize their 3D scene mapping to each other to ensure that virtual elements appear at a consistent position for every participant. Finally, participants have to collaborate with each other considering specific rules depending on the concrete application.

Now that the motivation and main idea is depicted, Section 1.2 specifies the problem, this work is going to face, in detail.

1.2 Problem Statement

Augmented Reality includes a wide range of different approaches augmenting a real world scene with virtual objects. To prevent confusion for the reader from the beginning, this chapter will start with a distinction of this work's approach to other common approaches. Section 2.1 will go into further technical detail.

As already mentioned, this work discusses an AR approach integrating multiple devices into a single AR scene. Each AR device is considered to be used by a different single user. This is not to be confused with works discussing collaboration in AR scenes visualized by a single screen or projector for all users. All of those users are supposed to be located in the same room, augment-

ing the same location, that means they are "co-located", in distinction to approaches of remote multi-user AR. Using AR with remote users offers a huge variety of additional advantages, like remote coaching or augmented conferences. At the same time, they face a different collection of challenges. For example, synchronizing visuals in relation to the real world may not be an issue for remote AR due to the fact, that in remote scenarios users don't see the same room anyway, or just share their views with each other. For orientating virtual objects at the real world, there are different technical approaches. In distinction to marker-based AR, this work will discuss so-called "markerless" or "location-based" AR. Marker-based AR uses recognizable images placed in the real world. If those markers are recognized, they are used as anchors to place virtual objects in relation to them. Markerless AR on the other hand uses other techniques. A simple approach to enable location-based AR is to use GPS to detect the device's position. In distinction to GPS, this work discusses surface detection. Anchors to orientate virtual objects at are placed at distinct positions on detected surfaces.

Approaches for AR applications integrating multiple devices circulate under several terms, like "multi-user", "collaborative" and "multi-device". As games are also a big area of application, the term "multi-player" is quite common, too. In the following, this proposal will use the term "multi-user" as description. The term "cross-platform" will be used to describe approaches integrating differing systems, most of all hardware and operating systems with each other, for example Android and iOS devices.

Now that the AR approach for this work is delimited, a concrete research question can be defined. The thesis' goal is a multi-user AR framework, not only providing a shared AR scene, but also management, communication and synchronization of interactions, rules and model data. There is a wide range of SDKs enabling and supporting developers to create applications featuring Augmented Reality on suitable devices. As a major part of those devices are handhelds, especially Google and Apple are leading the way in AR development. Apple provides the ARKit SDK[1] for developing AR applications for iOS devices. Google's equivalent to this is the ARCore SDK[2], which is not only available for Android devices, but also for iOS devices, supporting essential features concerning this thesis. To make multiple users take part in the same AR experience, Google has developed the concept of so-called Cloud Anchors, explained in detail in Section 2.5.

Not only visual synchronization is required, but also synchronization of meta data, managing business logic in a collaborative multi-device scenario and enabling user interactions with the AR scene. Figure 1.2 depicts an exemplary use case from the Mill app, in which a single user taps on a specific game piece on the board within the scene. By dragging, this object can be moved on the board to an adjacent free tile. All other participants have to realize the scene modification in real time. While a single user has the permission for this interaction, all other users' actions have to be restricted to keep the scene consistent. Without rules, every participant could transform virtual elements at any time. Collaborative use cases, like games, however, give users permissions to act in a specific way according to their rules. So, the challenge will be to combine the aspect of consistent visuals with the aspect of following and communicating consistent permissions and states as part of a business logic. This work is going to implement those features in a framework, that works as a generic foundation for a variety of domain-specific multi-user AR applications. Purpose of of those multi-user AR applications is of course collaboration in a shared AR scenario. This implies interaction with the AR scene, so that users are actively involved in

[1] ARKit | Apple Developer Documentation - https://developer.apple.com/documentation/arkit
[2] ARCore - Google Developers - https://developers.google.com/ar

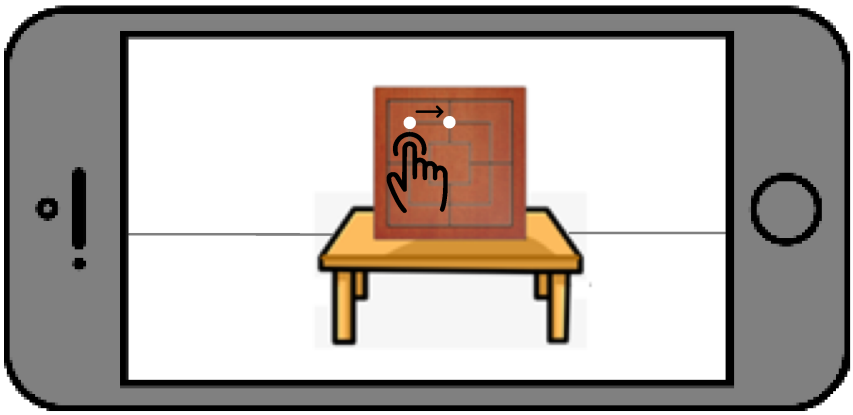

Figure 1.2: Interaction with the virtual mill board: Move a game piece to an adjacent tile via drag and drop.

the process and not only passive observers. Therefore, the main research question of this work is:

How to develop a cross-platform multi-user AR framework, that can be used for sample applications, which support collaboration with high usability.

The goal to answer the research question was to design, implement and evaluate such a cross-platform multi-user AR framework, partially integrating third-party components. In the following, the main challenges are listed, which came up during the developing process.

- **C1: Cross-platform AR engine**
  First an AR engine has to be researched and used, that enables compiling for a range of systems as wide as possible. In the best case, this includes iOS, Android and HoloLens support.

- **C2: Synchronized 3D scene**
  Like in any 3D multiplayer game, all participants must have access to the same synchronized 3D scene.

- **C3: Synchronized markerless AR scene including 3D scene**
  What is special about this work's multi-user AR approach, is that the 3D scene does not only require to be synchronous within itself, but also requires to be oriented in the real world in a consistent way through all AR devices' views.

- **C4: Synchronized metadata**
  To apply more complex rules for scene manipulation within an AR application, there also needs to be a way to store, read and manipulate metadata.

- **C5: Business logic computing in multi-computer scenario**
  Within a session with multiple computers interacting with each other, there needs to be a strategy, which device takes responsibility for which parts of computing. This heavily includes executing business logic.

- **C6: Cross-platform interaction modalities**
  Not only should as many operating systems be integrated as possible to participate, but the

device types supported by those systems should also be integrated. There may be different concepts of interaction for different device types, especially in regard of interaction with 3D scene.

- **C7: Domain-unspecific framework character**
  The software project should act as unspecific foundation to build various domain-specific applications with their help. It should work as reusable framework.

1.3 Solution Overview

Main part of this thesis was to develop the framework "XshARe" to solve the stated problem. Figure 1.3 gives an overview over its structure. After research on IDEs suitable for AR development, Unity was chosen as main engine for the project, as it comes with a capable engine to meet challenge **C1**. The Unity engine is suitable for developing both interactive 2D and 3D scenes. The result of compiling can be a standalone application for various platforms, for example iOS devices, Android devices and the Microsoft HoloLens. Section 2.4 goes into further detail on Unity and other IDEs. Unity 3D scenes can also be used as AR scenes using the device's camera recordings for room capturing. AR specific scene features can be addressed via Unity's AR Foundation interface. All applications based on the XshARe framework scan the room for horizontal surfaces. On top of tose surfaces an anchor can be placed. This anchor is the main landmark for all virtual 3D objects in the AR scene, which means, their 3D coordinates for positioning and rotation are managed relatively to the anchor's coordinates. The anchor is supposed to have same position and rotation in the room for all participating devices. Therefore, information about the scanned room and the anchor has to be shared between devices. This is done by an external cloud service of Microsoft Azure, the Spatial Anchors[3]. For setting an Anchor, data can be uploaded to the remote Microsoft Azure servers. Subsequently, all participating devices can download anchor data and use it to locate the anchor's position and rotation. The described process is executed by the Spatial Anchors example project provided by Microsoft. This example project for Unity was used as starting point for development and was subsequently expanded. This way the thesis did not have to reinvent the wheel of augmented reality, but could build up on a strong foundation already providing room scanning, 3D rendering and anchor sharing.

The framework is structured modularly. Its core consists of five main components expanding the Spatial Anchors foundation:

- **Spatial Anchors Coordinator**: This component is used to communicate with the Spatial Anchors third-party component, modify and expand its workflow. The Spatial Anchors are the foundation to meet challenge **C3**.

- **Input Control and Input Viewer**: To provide user interaction with the AR scene, this component reads out user input on the devices and interprets them into commands of interaction to manipulate the 3D scene and meta data. To fulfill the requirement to provide cross-platform support, all devices have to provide the same amount of interactions to meet challenge **C6**. The component Input Viewer enriches the AR scene with locally visible virtual objects to give input feedback.

[3]Spatial Anchors - https://azure.microsoft.com/en-us/services/spatial-anchors/

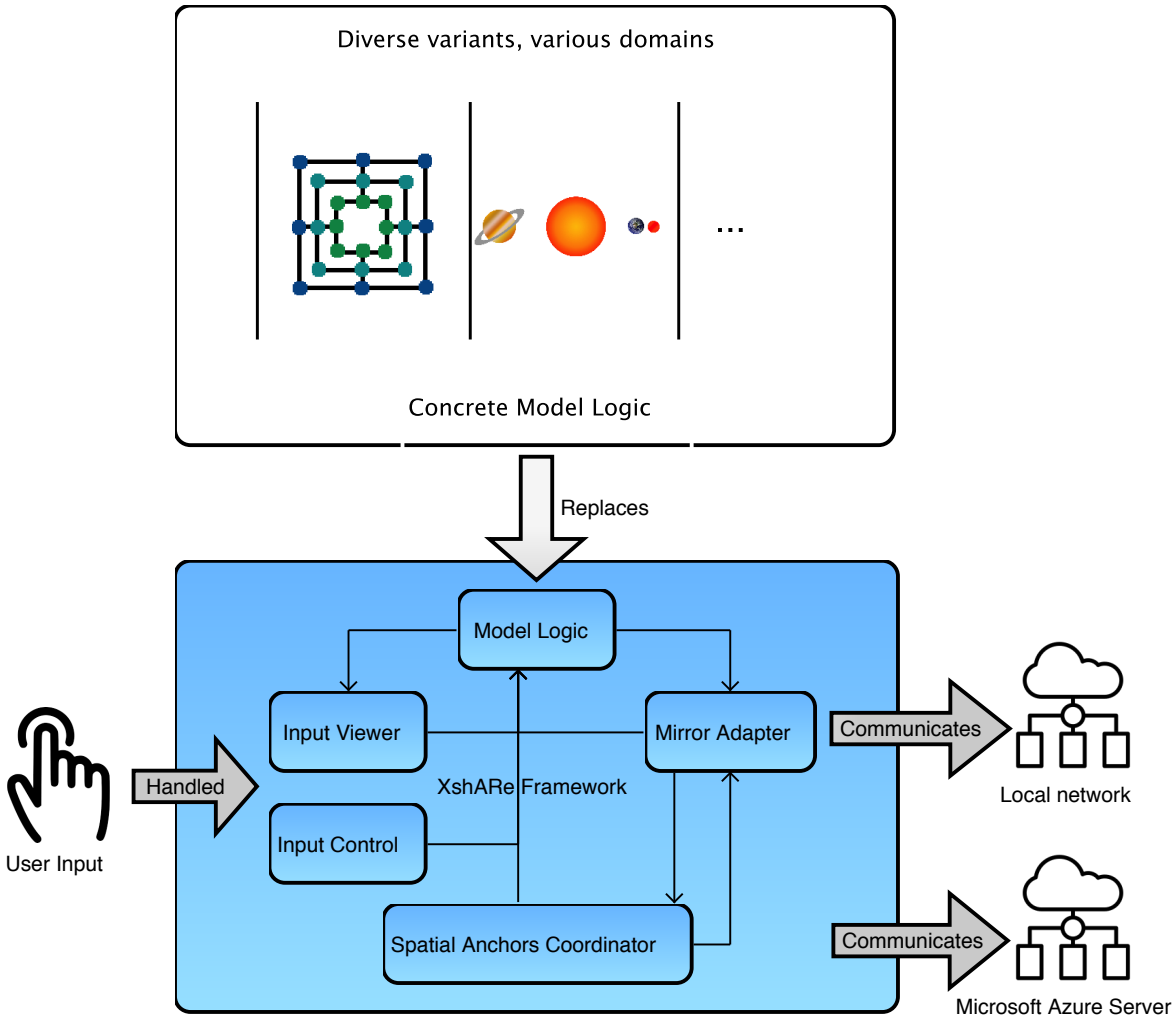

Figure 1.3: The developed solution: XshARe - a framework for cross-platform multi-user AR applications in Unity.

- **Mirror Adapter**: Mirror[4] is the used network component to synchronize data between connected devices. Further, server-run commands can be called by clients, if necessary. The Mirror network architecture is server-client-based. However, XshARe-applications do not require an additional dedicated server to run a session, as any participating device can act as host and therefore be a client and the server at the same time. Mirror is strictly server-authoritative. All data-changes have to be executed on server-side. Therefore, clients need to send server commands to commission a data manipulation. This data can concern visual objects' data as well as metadata (see **C2** and **C4**). Figure 1.4 illustrates the used client-server-based network approach. Section 2.6 discusses Mirror and other available network components for Unity in further detail.

- **Model Logic**: The fourth component is responsible for executing business logic as well as holding and accessing data. Business logic can be executed on client-side as well as on server-side (see **C5**). While all other components can be used without customization, the Model Logic component is the part, that needs to be replaced by an overriding concrete

[4]Mirror Networking - https://mirror-networking.com

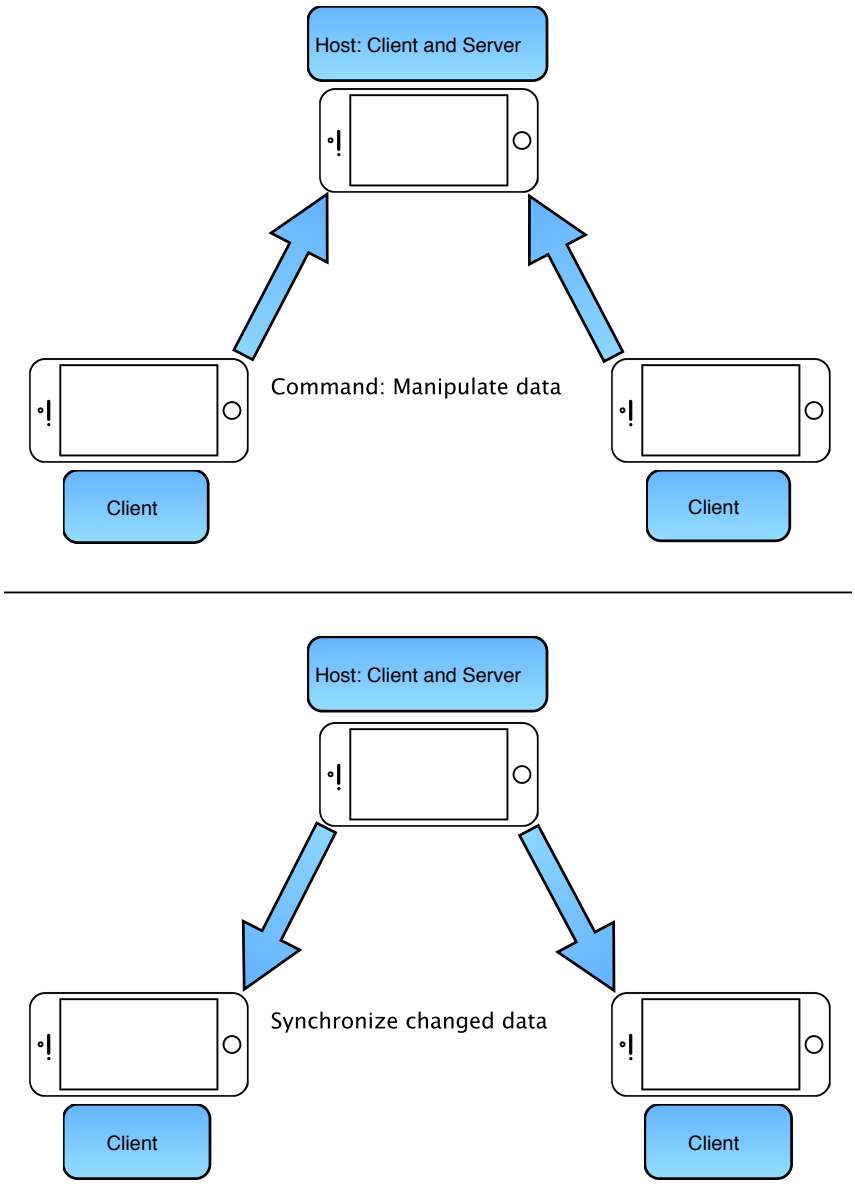

Figure 1.4: The used client-server approach to synchronize data and manage manipulations between multiple connected devices.

version for each individual domain-specific application (see **C7**). Unity holds visual data of virtual objects as so-called GameObjects, which can be extended by scripts for metadata and logic.

Chapters 4 and 5 go into further detail of these components.

1.4 Thesis Structure

Now that motivation, problem and solution are introduced, this thesis' following chapters go into further detail. Chapter 2 gives an overview about the work's foundations. It begins with a more detailed definition and explanation of Augmented reality. The following section then takes a look on a couple of currently available AR Software Development Kits (SDKs). AR Supporting

devices currently existing also get an overview. Subsequently, currently available IDEs suitable for AR development are examined. As there are technologies available for sharing AR anchors, an examination of them is also mandatory. The following chapter then examines and compare network components for Unity, the IDE of choice. At the end, an overview summarizes the results and chosen technologies to work with.

Chapter 3 gives an overview over found approaches of other scientific papers, which also discuss cross-platform multi-user AR. It takes a look on the features, stated challenges and stated requirements by the other approaches and outline the special features of the developed framework compared to others.

After research of related work is done, Chapter 4 discusses the framework's architecture and conceptual solution. To do so, use cases and requirements are analyzed. Design of component architecture and control flow follow up.

Chapter 5 discusses the implementation in detail. As the inner structure of developed components partially depend on the working methods of used third-party components as well as on the working methods of Unity itself, this chapter discusses structure and workflow inside components in detail. This includes explanation of technical specialties. Not only is the framework discussed in detail, but also two developed example applications based on the XshARe framework.

One of those two example applications is the AR Mill game already mentioned in Section 1.1. This application is subject for the project evaluation in Chapter 6. First the application is evaluated in regards of the requirements stated in Section 4.3. Subsequently, the application is also evaluated in terms of usability.

Chapter 7 then summarizes the results and gives an outlook on possible future work.

2 Foundations

To understand the developed solution, it is necessary to understand the basic foundations of AR in general, as well as the state of the art of modern available AR technologies. As AR technologies have evolved rapidly in the past years, it is quite likely to assume, that this work's actuality may fade within only a few years. It can be seen as a contemporary testimony for the current state of research and technology in AR.

2.1 Augmented Reality

The main idea of augmenting the visual real world with artificial elements is already many decades old. A very basic approach of this concept can already be found in 1901. Martinez et al. report their oldest found example of rudimentary augmented reality written down this date[MSH+14]. The idea includes augmenting persons with the visualization of personal metadata during a spectacular show. The development of computers and enhancement of computer graphics supported the evolvement of vague AR concepts. During the 1990s a more and more idea of AR was developed. At the same time, related concepts of mixing reality and virtuality emerged. Rekimoto and Nagao identified different styles of Human-Computer Interactions (HCI), as depicted in Figure 2.1[RN95].

The depicted scenarios of HCI are composed of atomic interactions classified as "Human - Computer", "Human - Real World" and "Real World - Computer". Four different types are distinguished:

- **GUI:** The Graphical user interface (GUI) is the regular style of interaction when working with a standard computer. The user can either interact with the computer or with the real world. Real world and computer are separated by a gap.
- **Virtual Reality:** In contrast to this style, Virtual Reality gives the user the feeling of being completely surrounded by the virtual computer world. Interaction with the computer world is easier, but in return, interaction with the real world is harder or even impossible, as the user can only see the virtual scene and the real world vanishes.
- **Ubiquitous Computers:** An alternative approach is to place multiple computers and displays throughout the room. Each of those displays could visualize a different virtual element and thereby create a mixed scene of real and virtual elements. Placing and moving those devices could however be complicated. The GUI gap may be smaller.

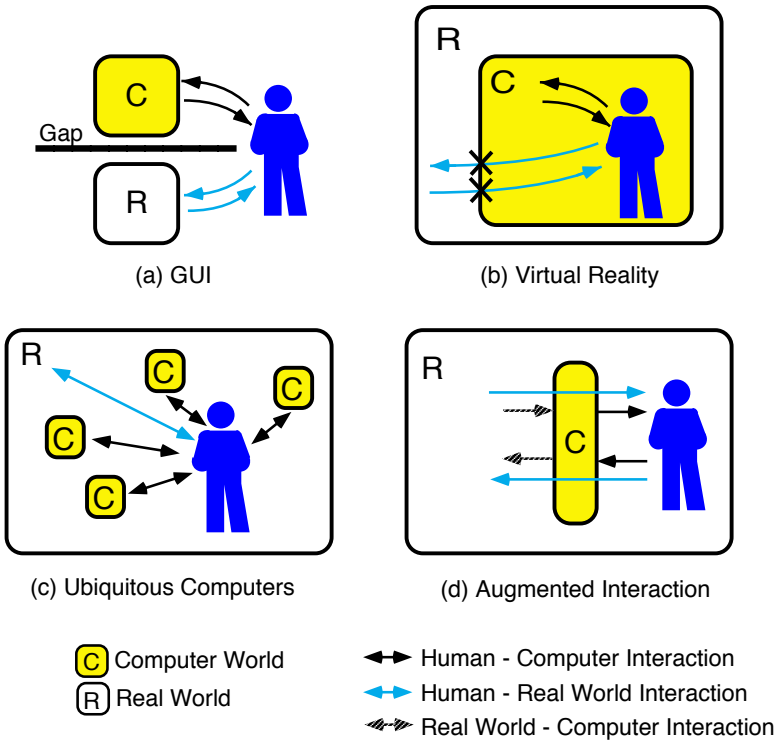

Figure 2.1: Styles of Human-Computer Interactions (HCI) - Source: Rekimoto and Nagao, 1995 [RN95].

- **Augmented Interaction:** Finally, the classifcation mentions Augmented Interaction, which is the closest to this thesis's used definition of AR. Users can see and interact with the real world without any complications through a computer display. Human-Computer Interactions are integrated via this display. Figure 2.1 also depicts an interaction between the computer and the real world. Schraffenberger and Van der Heide explore ideas for AR to have an impact on the real world via AR interaction or the other way round [SVdH13]. This work, however, does not consider those ideas for its AR approach.

Azuma describes AR as a variation of Virtual Environments (VE), a different term for Virtual Reality. Seeing the real world is described as the main difference between AR and VR. Therefore, AR is considered to be "the middle ground" between VE (completely synthetic) and telepresence (completely real) [Azu97]. Azuma emphasizes, that his definition of AR does not necessarily require the use of a head mounted display (HMD) or other specific technical device. This makes the definition quite practical for nowadays AR-related works, as AR supporting handhelds are more common than HMDs. Instead of considering used devices, the definition focuses on three characteristics:

- Combines real and virtual
- Interactive in real time
- Registered in 3D

Billinghurst et al. argue, that further technical requirements for AR can be derived from those three postulates. Without a display, real world and virtual objects cannot be combined.

Interactive graphics require a computer. As those graphics need to be visualized at a fixed position in the real world, a tracking system is required[BCL15].

Milgram and Kishino classified AR within a scale called the Virtuality Continuum. Figure 2.2 depicts the Virtuality Continuum. The range is divided into four parts: Real Environment, Augmented Reality, Augmented Virtuality and Virtual Reality. This fits with Azuma's definition, as AR is considered as a mixed form between real world and VR. Strictly speaking, AR is defined as a real world scene extended with several virtual objects, while Augmented Virtuality mainly is a VR scene with several real world objects. Both concepts are summarized as Mixed Reality.

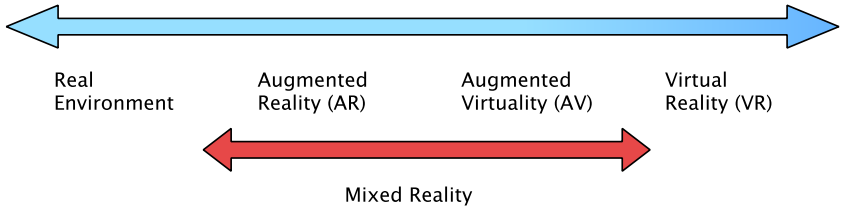

Figure 2.2: The Virtuality Continuum - adapted by Milgram and Kishino, 1994[MK94].

For modern AR applications, several different concepts are common to place virtual objects into the real world. Paladini distinguishes the following concepts in his blog article[Pal18]:

- **Marker-based AR:** In this variant, virtual objects are placed relatively to an anchor within the real world. This anchor is defined by a trackable image, which can be recognized by the AR application. The trackable image is called "marker". A special variant of marker-based AR is Image Augmentation, in which a virtual variant of the tracked image is placed directly on top.
- **Marker-less AR:** The easiest way to perform AR without marker is to put virtual objects just directly in front of the user's display. If orienting virtual objects to an anchor position is required, this can also be enabled by Surface Detection. The device can scan the room for horizontal or vertical surfaces. Anchors can then be placed at a position on those surfaces. This thesis is using such a marker-less anchor-based concept.
- **Location-based AR:** Some AR use cases also require considering the user's real world position. A considerable example would be an AR pathfinding application with virtual arrows showing the way to a target location. To get information about the user's position, it is common to query GPS data via smartphone.

2.2 AR supporting Devices and their Specifics

After the overview over AR, the next question to discuss now is what are requirements for AR-supporting hardware and what are nowadays common AR-supporting devices. As this thesis' project only considers handhelds and HMDs, uncommon device types like eyeglasses or contact lenses are ignored. Klepper discusses requirements for AR devices, especially in contrast to VR devices[Kle07]. Three required main subsystems are mentioned:

- Scene generator

- Display device
- Tracking and sensing

According to Klepper, the requirements for scene generators and display devices are lower for AR devices than for VR devices, as virtual objects in VR scenes have a higher need to look real in order to give the user an immersive feeling. Requirements for Tracking and sensing, however, are higher for AR devices, due to the need to have virtual objects placed in the real world in a stable way. Out of the variety of device types, this chapter focuses on three variants:

- **HUDs:** Head-up displays (HUDs) are fixed in front of the user's face. A HUD's content can only be seen by focussing the view on its display. HUDs are commonly used by pilots in aircraft. They consist of three parts: a computer, an overhead projector unit to project information and a combiner to reflect it in front of the user.
- **HMDs:** Head-mounted displays (HMDs), in contrast, occupy a greater field of view, so that their visualization cannot be overseen. HMDs work with a modulated light source with drive electronics viewed through an optical system (mirrors and combiner) which, combined with a housing, are worn on a users head via a headband, helmet, or around an eyeglasses frame."[RC05] An example for an HMD is the Microsoft HoloLens. As input modalities, the HoloLens provides a combination of gaze, gesture and voice inputs. The user's gaze can be used to focus on locations and objects in the field of view. Via gesture, a tap can be executed on the gaze focus. The HoloLens runs with Windows 10 as operating system.
- **Handhelds:** Handhelds can be classified in two main variants, that are commonly used nowadays: pads and smartphones. Most common operating systems for handhelds are iOS for Apple devices and Android by Google. These companies also provide free-to-use software to support AR on their devices. Section 2.3 elaborates on that.

2.3 Current AR SDKs and their Features

Software Development Kits (SDKs) with focus on Augmented Reality have evolved with rapid progress in the recent past years. SDKs are collections of tools and libraries for software developers. Instead of "reinventing the (square) wheel" by developing AR applications from zero (which is theoretically possible), relying on an existing AR SDK will very likely lead to more qualitative results. In his blog article, Gosch has listed remarkable AR SDKs to this date [Gos19]. The article is from 2019 and can therefore be considered actual. The following list is extended by several additional AR SDKs rated mentionable.

- **ARKit:** ARKit is the AR SDK developed by apple. It can exclusively be used with iOS systems. Therefore, using ARKit is a preferable choice in case developing exclusively for iPhones and/or iPads. ARKit provides high quality of tracking variants: surface detection, image tracking and even 3D-object tracking. The most mentionable ability of ARKit in regard of this thesis is definitively the approach for collaborative multi-user sessions. This collaboration can be implemented without the help of additional external cloud services. Section 2.5 elaborates on the topic. Minarik has used ARKit as foundation for a multi-user AR application, as documented in Chapter 3 [MJ19]. Due to the exclusive restriction on iOS devices, ARKit does not fulfill this work's cross-platform requirement. The IDE Xcode is required to debug and deploy iOS apps.

- **ARCore:** ARCore can be seen as Google's pendant to Apple's ARKit. Though it is mainly intended to be used on Android devices, ARCore can also be used as Wrapper around ARKit. This way ARCore enables the possibility to develop iOS apps. The developer can build the same AR app both for Android and iOS. An additional advantage of ARCore is the strong environmental understanding, especially in terms of lighting. Via light estimation, ARCore apps can calculate a realistic lighting for virtual objects in the AR scene depending on their position in the room and the lighting of the real world.

- **AR Foundation[1]:** AR Foundation is developed by the team of the Unity IDE and intended for the use with Unity. The SDK functions as wrapper for ARCore as well as for ARKit. The HMDs Microsoft HoloLens and Magic Leap are supported, too. AR Foundation is the AR SDK of choice for this thesis, as it provides a huge range of cross-platform compatibility and all of the required modern AR features. This includes surface detection as tracking concept to place virtual objects at a stable position. AR Foundation also provides the use of raycasts on trackable surfaces and objects. Those raycasts have their origin from the device's position. with regard to a 2D coordinate on the display, the raycast is shot into the scene and ends at a virtual or detected real world object. Selection and manipulation of virtual objects can be implemented this way. Figure 2.4 depicts a raycast. Section 2.4 elaborates on the use of Unity in further detail.

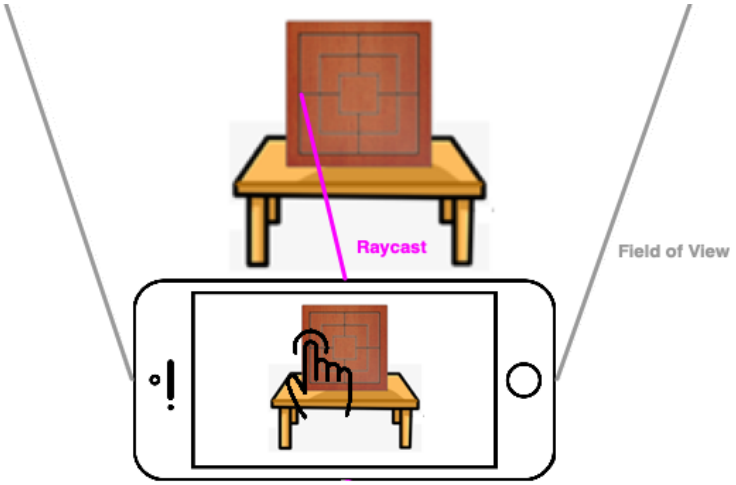

Figure 2.3: A raycast in Unity.

- **UnrealAR[2]:** The IDE Unreal also provides an AR template to use for developing AR applications. It also works as wrapper for ARKit and ARCore. UnrealAR can therefore be seen as Unreal pendant of Unity's AR Foundation.

- **Vuforia[3]:** Before tech giants like Apple and Google started to dominate the field of AR SDKs, there were already products available, which were developed by smaller companies. One of those still available SDKs is Vuforia. Vuforia is traditionally known for stable image tracking. Meanwhile, Vuforia also includes wrappers for ARKit and ARCore.

[1] About AR Foundation - https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.0/manual/index.html
[2] Unreal Engine - Augmented Reality Development - https://docs.unrealengine.com/en-US/Platforms/AR/index.html
[3] vuforia developer library - https://library.vuforia.com/getting-started/overview.html

- **Wikitude**[4]**:** Another AR SDK available, that came up first with several innovations in AR, is Wikitude. Wikitude was used to build the first location-based AR app[Per08]. It also was the first SDK providing full object recognition as further development of image tracking and surface detection.

- **A-Frame:** An obvious approach to cross-platform compatibility would be a web-based solution, as script languages like JavaScript and HTML5 can be interpreted by all modern-day web browsers on any device. A-Frame is a framework to enable web-based AR, but innately only marker-based and location-based with GPS detection. Placing virtual objects on a surface can be achieved by integration with ARCore. Google has developed the experimental AR web browsers WebARonARCore for Android and WebAROnARKit for iOS devices to run A-Frame-based WebAR with the surface detection of ARCore[Sha18]. The javascript-based web code defines the 3D models to be visualized as well as their pose. Those informations could easily be shared with other users. Network communication with A-Frame can be realized via WebRTC, as de la Puente shows in a blog article concerning a multi-user VR project[dlP17]. Sadly, the mapping information for surface detection would be captured by ARCore and would therefore not easily be accessible by a web script. Furthermore, only a small number of devices officially supports WebARonARCore. That is why A-Frame is not listed as a further approach to implement cross-platform multi-user AR.

2.4 IDEs and Engines suitable for AR Development and cross-platform Portability

Developing AR applications does not only profit from AR-specific SDKs, but also from relying on AR-supporting engines and IDEs. Integrated development environments strive for integrating multiple tools and functions needed within the programming process to reduce the user's necessity to switch between applications. Mostly these include editor and compiler. In case of AR application development, deployment on the end-devices is another important IDE feature.

Nebeling and Speicher examined and classified the wide range of AR and VR authoring tools[NS18]. Found tools are classified in 5 groups. Those groups differ in terms of fidelity on the one hand, skill and required resources on the other. In this context, fidelity refers to graphical quality of visuals. For example, required resources can include 3D models and hardware requirements, while skills mostly mean programming or 3D design skills. The optimal authoring tool would provide high fidelity, require few resources and skills. However, no authoring tool was found, that completely satisfied the authors. One identified class of authoring tools are platforms for game and application development. These are characterized as including a visual editor and integrating assets within a project. Those assets can for example be 3D models, textures, sounds, but also code scripts for process logic. The skill requirements constitute a barrier for non-technical designers. Mentioned examples were A-Frame, Unreal and Unity. As A-Frame was already discussed and excluded in Section 2.2, the focus will lie on Unreal and Unity in the following. Gajsek compared up-to-date versions of the two 3D engines[Gaj20]. Both are widely used for game development:

- **Unreal:** The Unreal engine already exists since 1998. To this date, it supports 15 different platforms to build applications for. Compared to Unity, it is known to have a higher focus on 3D graphics and higher quality of graphics. Therefore, it might be the engine of choice

[4] wikitude - https://www.wikitude.com

for VR applications. Due to the higher focus on graphics, the community includes more non-programmers. Unreal Engine 4 is open source since 2015.

- **Unity** The Unity3D engine was first released in 2005. To this date, it supports 28 platforms to build applications for, including all platforms this work has focus on. Supporting more platforms is definitely an advantage for a project, that aims at cross-platform support. Unity is widely known for being heavily used for mobile games, which also fits into this work's requirements. Unity provides a bigger range of components in the asset store, which implies a higher chance of finding useful third-party-components. Section 2.6 gives an example with the used network component. All of those arguments lead to choosing Unity as IDE of choice for this work's software project. Using platforms for game and application development can be combined with the use of additional tools. IDEs like Visual Studio can be used for coding and 3D design tools like Blender for modeling 3D models. Both of those tools were used in combination with Unity. Furthermore, Xcode was used for compiling and deployment on iOS devices.

  Unity projects consist of one or multiple scenes. A scene represents the room, in which all virtual objects are rendered. Scenes include lighting elements as well as camera elements. One camera is always the currently active one, that means, the image is rendered from the camera's perspective. In the same way, one scene is always the active one. A Unity application can switch from one scene to another. This is valid for AR scenes as well as for any other ordinary 3D scene. The virtual objects within scenes are represented as GameObject elements. Initialized GameObjects are clones of blueprints, that have to be designed and saved as assets within the project. Those assets can either be designed by an external 3D modeling tool like Blender, or composed of basic 3D objects within Unity itself. This means GameObjects can consist of further GameObjects. This leads to a hierarchical structure. GameObjects are linked to their parent and children via a Transform object. Transforms also reference their GameObject's 3D pose information. This includes position, rotation and scale. Global pose information defines the coordination within the scene overall. Furthermore, GameObjects also hold local pose information, which defines the coordination relatively to the parent game object. If there is no parent GameObject, local and global pose information are identical. Position and scale are held within Vector3 elements. Those mainly consist of three numeric values: The x, the y and the z coordinate. Rotation information, however, is stored in a Quaternion element, which consists of four numeric values between zero and one. The model of a Unity scene can completely be dynamically accessed and modified via MonoBehaviour scripts. Those scripts are one of the many types of Components, that can be attached at GameObjects. Figure 2.4 depicts the hierarchical concept of GameObjects.

2.5 Cloud Computing and current multi-user AR-related Cloud Services

Multi-user AR requires a synchronized AR scene. This does not only imply a synchronized 3D scene, but also a consistent view between all participating devices, so that virtual objects are positioned at the same real world location on all displays. As virtual objects can be attached to anchors in AR scenes, an anchor synchronization is the first step to a synchronized AR scene. Synchronizing anchors does not mean synchronizing their 3D coordinates within the scene, but rather detecting a common location in the real world. As already mentioned, Apple ARKit is an AR SDK providing this feature. Sadly, ARKit only supports iOS devices and can therefore

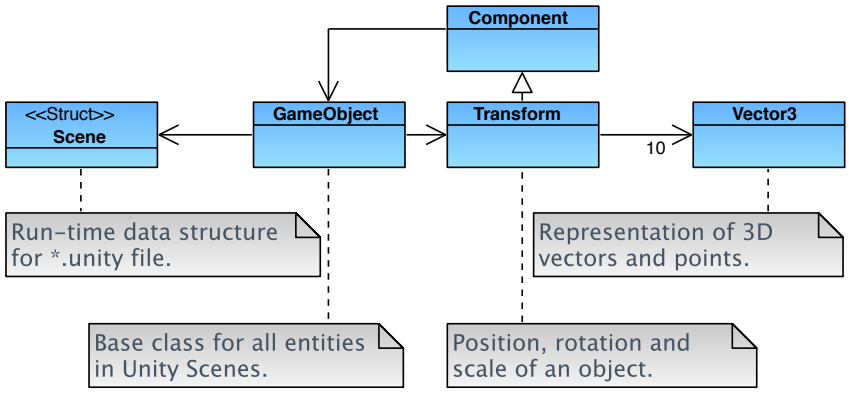

Figure 2.4: The hierarchical concept of GameObjects in Unity.

not be considered cross-platform compatible. Other approaches use Cloud services via internet, that provide the sharing of room scanning information as well as anchor detection for all participating AR applications.

The National Institute of Standards and Technology (NIST) has published a widely acclaimed definition of cloud computing. "Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." [MG+11] The definition lists five essential characteristics of cloud computing:

- **On-demand self-service:** Computing capabilities can be provisioned by the consumer automatically and without further human interaction required. That means, the interaction is executed between a client-side application or script and the cloud service provider.
- **Broad network access:** The service can be used by a diverse range of client devices. These can be smart phones as well as tablets, laptops or workstations. They can access the service over a network. In the following, this work will focus on internet-based services only.
- **Resource pooling:** There is a single pool of resources for all possible consumers. Required resources are assigned dynamically to consumers on demand. That means, the consumer has not knowledge or control over the exact location of used resources.
- **Rapid elasticity:** As resources can be provisioned and released dynamically, the capabilities might appear unlimited to the single consumer.
- **Measured service:** The service provider can monitor, control and report the requested services and the used resources. This provides transparency for the provider as well as consumers

Cloud computing concepts can be classified by several different service models. These service models differ in the kind of actual service, they provide and can be seen as building up on each other:

- **Infrastructure as a Service (IaaS):** The consumer can request fundamental computing resources such as processing, storage and networks. Those resources can be used to run

software on. Types of software can be operating systems as well as applications. The computing resources are not controlled by the consumer.

- **Platform as a Service (PaaS):** Applications, that was created or acquired by the consumer can be deployed onto the cloud infrastructure. Those applications can then be run the remote operating system or runtime environment. In addition to IaaS, the consumer has no control over the runtime environment, either.

- **Software as a Service (SaaS):** The service also provides running specific applications. These applications can be interacted with via an application or script as interface on client-side. In this case, the consumer does not even control the application capabilities themself.

- **Function as a Service (FaaS):** A younger service model is Function as a Service, also known as "serverless architecture". This model hides the server-architecture from the consumer, like in PaaS. While in PaaS the application capabilities are still in control of the consumer, in FaaS the function to run is in control. The difference between the two models is, that PaaS always requires to have at least one server process to receive external requests, which is not required in FaaS [Avr16].

Several tech giants provide cloud computing platforms including various services, also computing services, that can be classified as SaaS. Two of those platforms are Google Cloud Platform and Microsoft Azure. Both provide a service for AR anchor sharing, which requires registration and could be used for free in relation to this work. Shared anchors are given different names by each of the providers. In Apple's ARKit they are called *ARAnchor*, while they are just called *Anchor* in Google's ARCore and *World Anchor* in context of Microsoft's HoloLens [Pap20]. During this thesis, the term of *shared anchors* is used for them to make clear their function.

- **Google ARCore with Cloud Anchors**[5]

  A cloud-based example of anchor-sharing are CloudAnchors from the Google Cloud Platform, an approach both compatible with Android and iOS devices. An anchor describes a specific position inside an AR scene, at which graphical objects can be oriented. There can be multiple anchors placed in a scene and multiple virtual objects oriented at an anchor. When the device's camera moves through the scene, only the anchors' position needs to be calculated relatively to the scanned 3D map. All pinned virtual objects stay constant to the anchor and therefore constant to each other, providing a more solid user experience. Anchor data can be uploaded to the ARCore Cloud Anchor service and from there be used by other users synchronizing their own AR scene. Figure 2.5 depicts the Cloud Anchor principle: Virtual objects are placed in a 3D scene with coordinates related to the anchor. The anchor itself is placed at a specific position on a surface in the scanned room. All data necessary to comprehend the scene is uploaded to Google's service cloud, which both participating devices have access to. This way both users share synchronous AR experience.

  Google's SDK ARCore claims to integrate Android and iOS systems and can therefore be called cross-platform supporting. However, this support is restricted on handhelds and Google's own smart glasses. Interaction differences between handheld and HMDs are not part of the kit. All in all, ARCore brings a good base in terms of features considering the multi-user coordination of AR data itself, but still leaves the developer with the further

[5] Share AR Experiences with Cloud Anchors - https://developers.google.com/ar/develop/java/cloud-anchors/overview-android

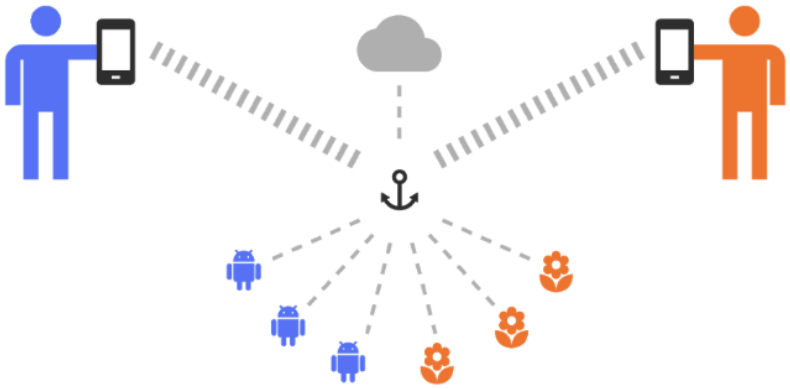

Figure 2.5: ARCore Cloud Anchor principle: Users share information about an anchor in the scene via server. Multiple virtual objects can be placed by all participants relatively to the anchor's position.

process and data management beyond.

- **Microsoft Azure Spatial Anchors**[6] The pendant from Microsoft Azure, called Spatial Anchors, works in a similar way. After registration at Microsoft Azure, a resource has to be defined via web interface. This resource is later related to the application to be run. The whole process of establishment is documented in a tutorial[7]. Microsoft provides an example project for Unity, that can be used as AR application and client requesting the cloud service. In addition, the project includes code for a web application. This web application can be uploaded to the Microsoft Azure platform via Visual Studio. This will be the software, that will be requested via internet, run on remote servers and execute the anchor sharing. The web application provides computing and saving information about multiple rooms and anchors. Each room is associated with a unique ID, beginning from 0 and ascending. Chapter 5.2 goes into further detail on the example project. Test runs showed data persistence for about an hour. Microsoft Azure claims, that persistence can be extended by combination with an additional database service. This way, persistent AR could possibly be enabled. Spatial Anchors are compatible with Unity's AR Foundation and claims to be usable on iOS and Android handhelds, as well as on the Microsoft HoloLens. To this date, the service is in preview phase and therefore free to use.

  Meeting stated requirements and integrating used third-party components has a huge impact on the solution's overall design. Especially using Spatial Anchors sets design prerequisites, as the Unity project Azure Spatial Anchors Examples is supposed to be used as foundation for the framework. The developed framework then serves as foundation for any domain specific AR applications built with it. The Azure Spatial Anchors Example project comes with two scripting packages. Package AzureSpatialAnchors.Examples includes scripts, that get directly attached to GameObject elements in Unity scenes. These scripts inherit from the class MonoBehaviour, a class provided by Unity to implement

[6]Spatial Anchors - https://azure.microsoft.com/en-us/services/spatial-anchors/
[7]Tutorial: Share spatial anchors across sessions and devices - https://docs.microsoft.com/en-gb/azure/spatial-anchors/tutorials/tutorial-share-anchors-across-devices?tabs=VS%2CAndroid

program logic. The second package is AzureSpatialAnchors.SDK. The MonoBehaviours in the first package call services from this SDK. AzureSpatialAnchors.SDK controls communication with the web application on Microsoft Azure server-side. It also accesses services from the AR Foundation SDK of Unity, which serves as wrapper for system-specific AR SDKs like ARKit for iOS devices or ARCore for Android devices. Figure 2.6 depicts the layered structure of this solution.

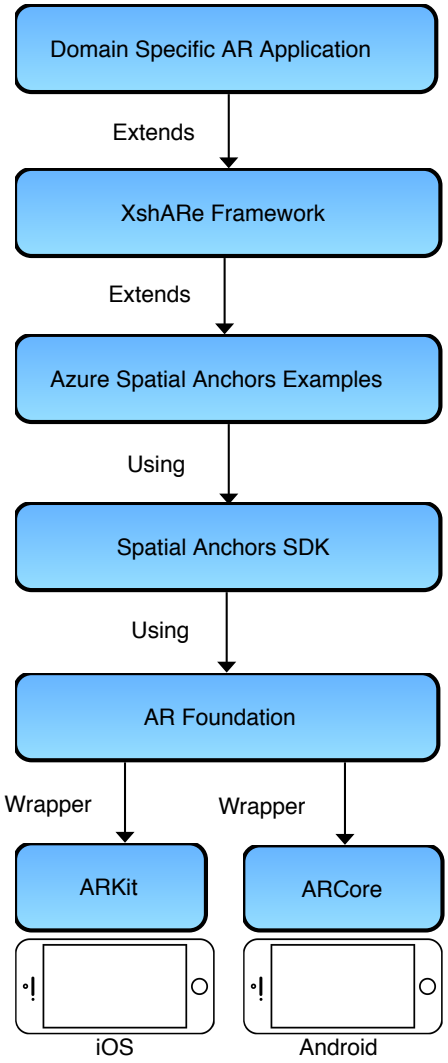

Figure 2.6: Layers of the XshARe framework.

Table 2.1 gives an overview over all examined provisions concerning sharing room and anchor information and their cross-platform support. As Microsoft Azure Spatial Anchors are the provision with the highest range of cross-platform support, it is the provision of choice for this thesis.

2.6 Computer Networks and third-party Network Components for Unity

Now that an approach is chosen to synchronize anchors with consistent real world positioning on all participating devices, virtual objects can be oriented at them. To view them with con-

	iOS	Android	HoloLens
ARKit	✓		
Cloud Anchors	✓	✓	
Spatial Anchors	✓	✓	✓

Table 2.1: Overview: cross-platform support of examined anchor sharing provisions.

sistent real world pose as well, their coordinates require to be treated relatively to the anchor coordinates. Those relative coordinates require to be held synchronous on all devices, as well as possible metadata. That means, the involved devices need to communicate with each other. This is done via a computer network. A computer network is a group of computers, that is able to digitally communicate with each other via protocols. A conceptual standardization model for communication functions in computer networks is the Open Systems Interconnection model (OSI model). It consists of seven layers[Bri00]:

- **Layer 1: Physical Layer**
  The lowest layer is concerned with transferring bits with the help of hardware tools.
- **Layer 2: Data Link Layer**
  On top of the physical layer, the second layer is responsible for avoiding transferring errors as well as accessing the tranferring physical medium.
- **Layer 3: Network Layer**
  The network layer is responsible for routing and transferring of data packages. Routers are working on this layer.
- **Layer 4: Transport Layer**
  Packages transferred on layer 3 are built on layer 4. The data stream gets segmented.
- **Layer 5: Session Layer**
  Layer 5 controls the process of communication.
- **Layer 6: Presentation Layer**
  The presentation layer is concerned with data formats. Application data is packed and unpacked. Furthermore, encryption and decryption takes place on this layer.
- **Layer 7: Application Layer**
  Finally, the last layer is the interface for applications to use a network via protocols for data input and output. All other layers are connected by this layer.

Computer networks can be modeled as graphs with nodes as connected computers and routers and edges as connections between those devices. The graph structure can result in different kinds of topologies, for example ring, star, tree or line. Connections can either be wired or wireless. This thesis is focussing on wireless WIFI connections in the following, as it is the only kind of network connection used. The WIFI network is provided by a router. As all network communication trespasses the router, the network topology is a star in this case. The network is used for distributed computing. There are several main architectures of distributed computing models[ATKS]. Figure 2.7 depicts those models.

- **Client-server:** A single computer acts as server. All other computers are clients able to send requests to the server. The server can then answer the request, but otherwise stays passive towards the clients.

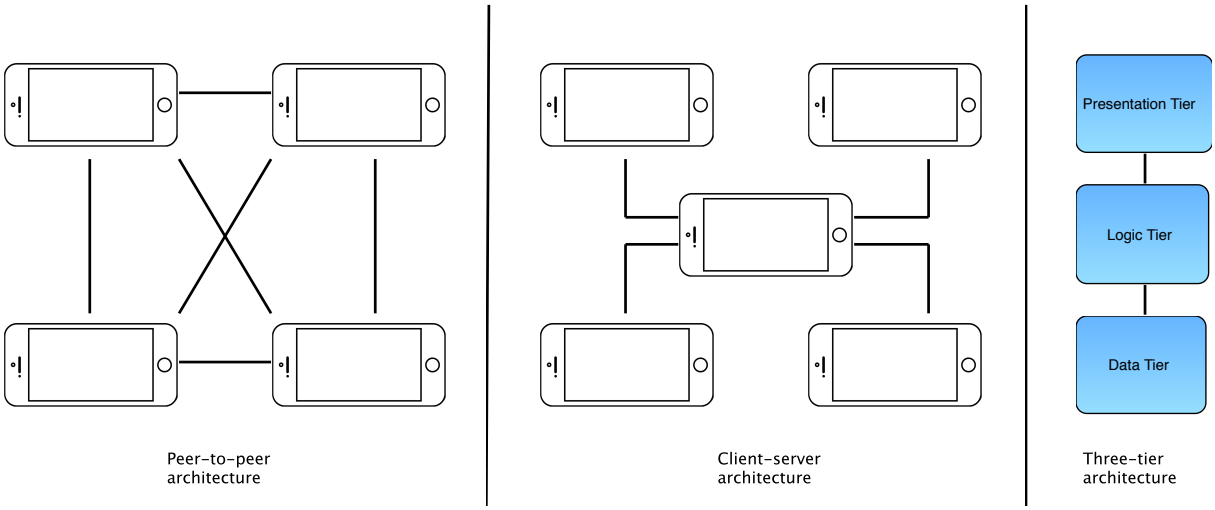

Figure 2.7: Models of distributed computing architectures.

- **Three-tier:** The three-tier model is commonly used in cases relating to web-applications. Three machines are involved in the application. The top tier is the presentation tier. This is the user interface running on the end device. Underneath lies the logic tier computing business logic on a remote computer. The last tier is the data tier responsible for data storage in a database or file system.

- **Peer-to-peer:** All connected computers, called peers, can act as servers as well as clients. This means, responsibilities are divided uniformly among the involved computers.

To enable network communication, this thesis' software project relies on a third-party network component. This way, the development avoids confrontations with detailed technical questions on networking. Even though Unity offers its own network component, called UNet, this can no longer be used for the project, as it is already deprecated and soon be replaced by a new network component with new API. The new network component will require using a dedicated server additional to the client devices [Glo18]. There are several third-party network components available for Unity. House compared the most popular ones in a blog article [Hou20]. Figure 2.8 summarizes the results for all provisions in regard of all considered aspects.

The most important aspect for this work's choice are the costs, as only free-to-use components can be considered. This leaves two provisions to choose from: MLAPI[8] and Mirror[9]. While MLAPI is reported to have better performance and wider feature breadth, Mirror is reported to be easier to use and have a helpful community. As this work does not have any non-functional requirements regarding network performance, the choice falls on Mirror. Its ease-of-use is a high benefit, regarding the author did not have any prior knowledge on the subject.

Mirror always uses a server-client architecture. However, one of the involved end devices can be host defined as host. In this case, the device acts as server and client at the same time. Mirror is strictly server-authoritative. That means, data can only be modified on server-side (see Figure 1.4). Therefore, clients need to call commands on the server for data modification. The Mirror API tries to resemble the UNet API as closely as possible. Listed below are the most

[8] What is the MLAPI? - https://mlapi.network
[9] Mirror Networking - https://mirror-networking.com

	Stability/ support	Ease-of-use	Perfor-mance	Scalability	Feature breadth	Cost*	Customers recommend for
MLAPI	★★★★★	★★★★☆	★★★★★	★★★★☆	★★★★★	Free	Most client-server games for up to ~64 players that want a stable breadth of mid-level features
DarkRift 2	★★★★★	★★★☆☆	★★★★★	★★★★★	★★★☆☆	$100 for source	Games with high perf/ scale requirements that want to build on a stable LL layer
Photon PUN	★★★★☆	★★★★★	★★★☆☆	★☆☆☆☆	★★★★☆	$0.30/PCU	Simple and small (2–8 players) mesh-topology games
Photon Quantum 2.0	★★★★☆	★★★★☆	★★★★★	★★☆☆☆	★★★★☆	$1000/mo + $0.50/PCU	Games desiring deterministic roll-back, like MOBA games, for up to 32 players
Mirror	★★★★★	★★★★★	★★★☆☆	★★★★☆	★★★★☆	Free	Stable and proven client-server solution, loved best for its community and ease-of-use

* Note that Photon pricing provides access to the networking libraries and services, whereas other solutions are standalone networking libraries, and the cost of services is separate.

Figure 2.8: Table: Comparison of third-party network components for Unity. - Source: House, 2020 [Hou20]

important components to use or consider in the project. Section 5.3 goes into further detail on using those elements.

- **Network Discovery:** Automatically find accessible hosts to connect with as client.
- **Network Identity:** This component is required to be attached at every GameObject, that is to be synchronized via network.
- **Network Manager**: The Network Manager is the main component to define networking details.
- **Network Manager HUD/Network Discovery HUD:** The graphical HUDs control connection establishment. They are useable for testing on desktop, but hardly usable on phone displays, due to their format. This is way they have to be replaced for AR applications.
- **Network Transform:** The Network Transform component is used to synchronize world coordinates of a GameObject.
- **Network Transform Child:** In contrast to the Network Transform, this component is used to synchronize local coordinates of a GameObject.
- **Network RigidBody:** Synchronize physics behavior like gravity and collisions.

2.7 Overview of chosen Technologies, their Advantages and their Scope

Now that foundations on used technologies and concepts are discussed, a short overview follows on the choices made to accomplish this thesis' software project of developing a cross-platform multi-user AR framework.

- **Supported devices:**
  Section 2.2 discussed AR supporting device types. The framework is going to support android and iOS handhelds. With the Microsoft HoloLens, an optional support for an HMD is also pursued. AR on iOS devices is enabled via the ARKit SDK provided by Apple, while Google provides the ARCore SDK to enable AR on Android devices.

- **Used AR SDK: AR Foundation**
  Section 2.3 presented current AR SDKs and their features. Due to the cross-platform demand, AR Foundation is the optimal SDK to rely on, as it acts as generalizing wrapper for various AR SDKs, especially for ARKit and ARCore. AR Foundation is the standard AR SDK provided by Unity.

- **Used engine and IDE: Unity**
  IDEs and engines suitable for AR development and cross-platform portability are discussed in Section 2.4. Unity is the inevitable consequence out of the commitment to AR Foundation, as it is inherent part of Unity. The IDE provides an editor to design 3D scenes including AR scenes. Virtual objects represented by so-called GameObjects can be enriched with code scripts to implement dynamic program logic.

- **Used AR anchor cloud service: Microsoft Azure Spatial Anchors**
  Section 2.5 investigates cloud computing concepts and currently available cloud services for anchor sharing. The Spatial Anchors provide the widest range of cross-platform support (iOS, Android, HoloLens) and are therefore the cloud service of choice for this project. Mapping and anchor data can be uploaded and downloaded on Microsoft Azure servers via a web application

- **Used third-party network component: Mirror Networking**
  Currently available third-party network components are discussed in Section 2.6. Mirror is the network component of choice due to its intuitive, stable, easy and free use. The component only works with a client-server architecture. A device can register itself as host and then act as both client and server.

3 Related Work

Now that the foundations for this work are introduced, this chapter will take a look on related work. This includes concepts and approaches from scientific papers as well as researched software, that fits into the description of multi-user AR. Not only are those approaches described, they are also examined in regard of characteristics similar to this thesis' own developed framework. Section 3.1 derives those characteristics, which are requirements for a satisfying cross-platform multi-user AR framework. Section 3.2 then describes all examined approaches in detail. The chapter finishes with a summary over all approaches in regard of meeting the requirements. This is done in Section 3.3.

3.1 Requirements: Examined Characteristics

Before related work is discussed, characteristics to be examined have to be discussed. Section 1.2 introduced challenges faced during the developing process. Those challenges, however, are partially to technical to act as requirements for a satisfying cross-platform multi-user AR framework. That is why there is a need to define a new set of characteristics on a more abstract level to examine related work. These characteristics are also required to enable the multi-user AR mill example presented in Chapter 1.

- **R1: Supporting multi-device sessions**
  Each participating user has his own device with an own display as well as camera. This is in contrast to approaches providing a shared screen for multiple users. Challenges **C1** to **C5** are all related to this requirement, as they all make sense within multi-device sessions exclusively.

- **R2: Supported cross-platform range**
  The supported cross-platform range is modeled as a set of systems able to run AR applications. In the best case, this set includes iOS, Android and Windows 10 for Microsoft HoloLens. This requirement is connected to challenge **C1**. While this work relies on a third-party AR-engine, other works may have developed their own.

- **R3: Supported device types**
  Connected to the platform range is the device range. An optimal case supports a set including handhelds as well as HMDs. Additionally to challenge **C1**, **C6** is also connected to this requirement, as different device types can support different interaction modalities.

- **R4: Consistent AR view**
  Holding the AR view consistent between devices is one of the core features of this work. Other approaches may just dispense such a feature or be solely remote. A consistent AR view requires a synchronized 3D scene as well as a consistent integration of this 3D scene into the real world. Challenges **C2** and **C3** are related to that.

- **R5: Co-located AR**
  Approaches may be exclusively remote, co-located or combine remote and co-located AR. The last two variants will be evaluated as satisfying. Co-located AR is related to challenge **C3**, as the distinction between marker-based and marker-less AR is only useful in co-located scenarios.

- **R6: Markerless AR**
  Approaches may be using either marker-based or marker-less AR. Only marker-less approaches are evaluated as meeting the requirement.

- **R7: Interactivity for multiple users**
  This requirement is only met, if all participating users can actively interact with the AR scene via their individual device. In addition to challenge **C6**, it is also related to **C5**. This is because interactions cause processes on business logic layer and afterwards manipulation of data. This requires a strategy in regard of what to execute on which device.

- **R8: Framework character**
  This required characteristic is equal to challenge **C7**. Other approaches may be domain-specific standalone applications instead.

3.2 Survey: Examined Works

Various approaches on multi-user AR applications and frameworks can be found. Goal of this section is to describe those approaches in regard of differing characteristics, features and met requirements.

A1: XD-AR[SHY+18]
Nebeling describes the development of a cross-platform multi-user AR framework "XD-AR" integrating HoloLens, Tango and RoomAlive systems. The approach spares the use of external services and manages network communication via a server. The approach tries to shift as many processes as possible on server side, to have spare implementation effort for each platform. The stated requirements are focused on functionality. Usability is less focused. The implemented framework provides functionalities similar to modern AR SDKs like ARCore but supports different platforms.

XD-AR is an approach integrating HoloLens, Tango and RoomAlive applications and therefore claims cross-platform usage for handhelds, glasses and projective AR. Due to the variety of devices, it also resolves differing interaction modalities between the supported device types. However, it sets no focus on usability concerning a standard for easy connection and leading the mapping process, which can be complicated for unexperienced users. Considering user positions as further points of interest is completely ignored. Based on two scnenarios, there are 7 challenges listed to overcome by the XD-AR framework:

- Availability on different, potentially low-end, devices (similar to **C1**)

- Stable tracking and exact positioning (related to **C3**)
- User decision between marker-based and marker-less AR
- Supporting different displays
- Synchronized AR content (similar to **C3**)
- Supporting device-specific input modalities (similar to **C6**)
- Global coordinate system (similar to **C2**)

While some of the listed challenges have adequate pendants in this thesis' list of challenges, there are some differences. While XD-AR also supports marker-less AR, this work does not. Furthermore, this work does not consider any display differences to be challenging in the expectation, that the used AR engine Unity is responsible for rendering and display modalities. Management of metadata and business logic is not considered as a challenge. Another critical difference to this thesis is the approach to establish a global coordinate system. That means, that all virtual objects have the same global coordinates on all devices. Shared anchors are not used. As the approach includes a server-side application, this application is solely responsible for various computing parts, that are classified in three components: Coordinate Management, Communication Management and Session Management. The tasks assigned to those components are structured completely different than in this work, though. Figure 3.1 depicts the framework's structure.

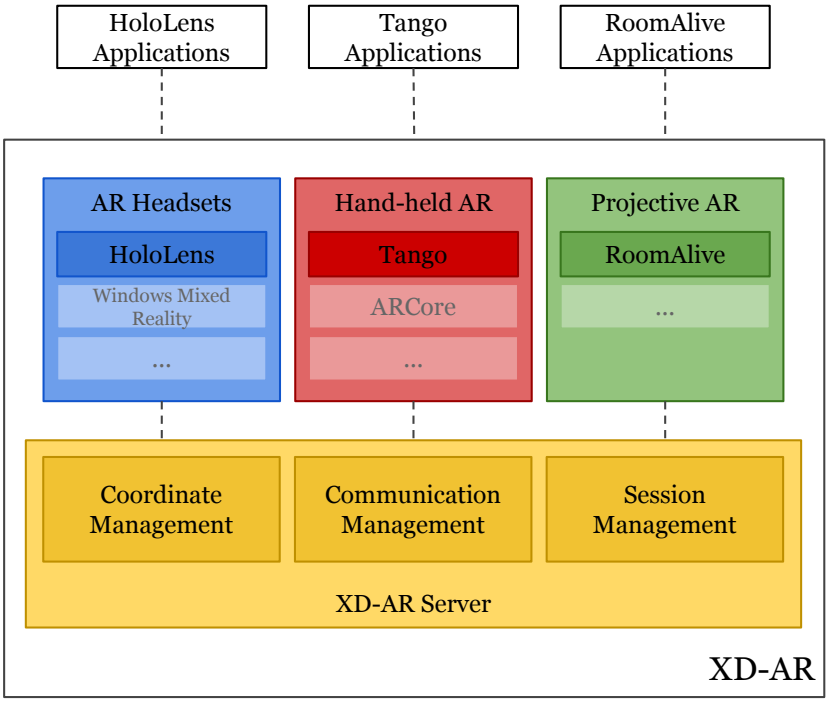

Figure 3.1: The XD-AR framework - Source: Nebeling et al., 2018[SHY+18].

While Communication Management is responsible for cross-device network communication, just like a network component, the session management manages the synchronization of object existence within a scene and ensure persistence. Modern third-party network components in Unity tend to include this feature. The coordinate management is responsible for possible

translation between different kinds of coordinate systems. This is a task not required when working with Unity, as all applications use the same format of coordinate system. Instead, an exclusive synchronization of local coordinates relative to the shared anchor's position is required.

Though, the framework XD-AR is very satisfying in regard of meeting stated requirements, it has two critical disadvantages to the newly developed framework. First, it is dependent on an external server running within the computer network. To make applications practical in everyday life, accessing the server via internet would be necessary. This would again require management of sessions possibly running simultaneously. A possible alternative would be to compute all logic on end devices. This approach, however, has the downside of being dependent on the end-devices' hardware resources and runtime capabilities, which are possibly lower than the ones of a stationary server. The second disadvantage is the requirement to program a unique software project for each supported platform. Relying on a single Unity project and compiling or exporting it for various platforms is a much more comfortable and time-saving approach for developers.

A2: Placenote SDK. [Mat18]
The Placenote SDK framework approach is heavily focused on usability and integrating AR coordination and coordinating all further shareable program information. Mathew describes an approach to comfortably build a multiplayer AR game for iOS devices with the advertised Placenote SDK [Mat18]. As a special feature, the article concerns the localization of each device relatively to each other. Placenote SDK only supports iOS devices to this date and therefore lacks cross-platform support.

Five components are stated to be necessary for a multi-user AR application:

- Network connection
- World mapping in own session
- Cloud-based world map sharing
- Communication of player position
- Metadata and and action synchronization

In the self-developed XshARe framework, components 2, 3 and 4 are workflow steps already implemented within the Spatial Anchors example project. Network connection is enabled by the third-party network component and integrated within the preparation workflow. The list only mentions communication of player positions, but lacks mentioning position synchronization of all virtual objects placed relatively to shared anchors. This is required as well as metadata synchronization.

A3: "Collaborative Augmented Reality" [Mim14]
Mimaroglu presents a framework to marker-based AR. The work was published in 2014. Due to his publication date, the approach is quite modest and cannot fall back on nowadays SDKs provided services. Instead, it uses the AR SDK of Vuforia. Cross-platform support is not mentioned directly, but the framework focuses on supporting only mobile devices and mentions Android and iOS as conceivable platforms. Glasses and their integration are ignored. Network communication is realized via Alljoyn, a communication framework for mobile devices. Figure 3.2 gives an overview over the framework's architecture.

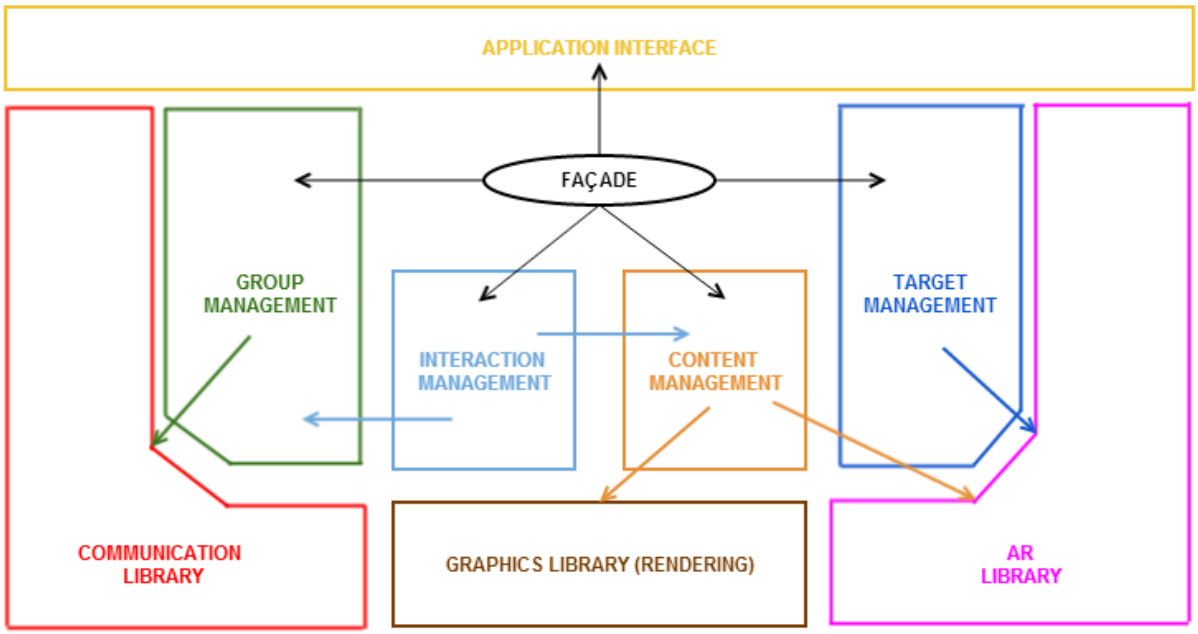

Figure 3.2: The framework archtitecture for "Collaborative Augmented Reality" - Source: Mimaroglu 2014 [Mim14].

The architecture can be divided in several layers. The lowest layer consists of libraries for basic tasks. Those libraries are accessed by management modules above. The connector for applications built on top of the framework is the component "Façade". Responsible for network communication are the Communication Library and the Group Management for managing sessions. Content Management and Graphics Library are responsible for rendering. The Interaction Management Module has influence both on graphics and communication modules. Target Management is responsible for marker tracking and interacts with the AR Library. The Façade module is a singular API and provides access for group and session registration, interaction as well as on content and target management. Device interaction modalities, metadata management and business logic have to be developed on top of this foundation.

A4: "A collaborative Augmented Reality framework based on Distributed Visual SLAM" [ET17]

Egodagamage developed a framework mainly focussing on transacting the mapping session for markerless AR. The work presents an approach to simultaneous locating and mapping (SLAM), a technique, that is also used by nowadays AR SDKs like ARCore and ARKit. The SLAM session is managed collaboratively, that means, multiple devices contribute image data from their positions to build an overall 3D map. Due to hardware resource restrictions, this framework does not run on handhelds or glasses. Instead, the approach runs on a stationary computer and uses multiple cameras for the SLAM session.

A5: Spatial[1]

A very advanced software using multi-user AR is Spatial. Spatial is a conference tool combining two features. The first one is starting a conference with users being located in other rooms. Those remote users are projected virtually as drawn avatars [Lee18]. The second one is to use a room as a shared monitor during the conference. This means, desktop items, like im-

[1]Spatial - Collaborate from anywhere in AR and VR - https://spatial.io

ages, texts or 3D models can be drawn into the room and shown to any participants at the same location. To do so, first a user has to scan the room. The collected information is then shared to other participants. Spatial claims to be "hardware agnostic", which means the broadest range of cross-platform compatibility of all examples. Implementations are available for HMDs like Microsoft HoloLens, as well as for handhelds and web browsers [Dic18]. Although Spatial meets most of the defined requirements, it is a single purpose software targeting a concrete domain of use. A general framework for further specification is not provided.

A6: "Multi-user interaction with 3D objects in Augmented Reality" [MJ19]

Minarik presents an approach to use Apple's ARKit for realizing a multi-user AR application. In the same way as the framework, that is to be developed, this work from 2019 relies heavily on the technical features offered by nowadays AR SDKs. The decision for ARKit and against Google Cloud Anchors or Microsoft Azure was led by the discovered possibility to work independently from any cloud-based services and therefore independently from an internet connection. The commitment to ARKit means a commitment on iOS devices and therefore no cross-platform compatibility. The work mentions though, that Microsoft Azure's multi-user AR technology Spatial Anchors can be integrated in ARKit and ARCore projects. Further, the approach developed a workflow to resolve user conflicts in case of multiple users trying to interact with the same virtual object at the same time. This requires synchronizing information about interaction permissions across all devices. Figure 3.3 depicts this concept.

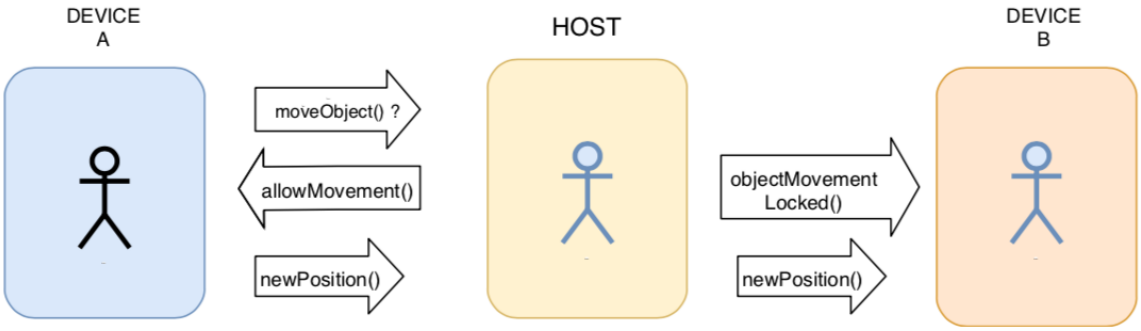

Figure 3.3: A concept for resolving user conflicts in multi-user AR sessions - Source: Minarik 2019 [MJ19].

Before an object can be manipulated, it has to be locked for all other participants. Example for data manipulation is the movement of a virtual object. What's special about this scenario, is that the object is moved via finger dragging, a process taking several moments. During this process, position data is actualized instantly. That means, object movement implies several data-write transactions by a single agent in a row.

A7: PROMAR [LNZ+19]

PROMAR stands for "Practical Reference Object-based Multi-user Augmented Reality". The project takes a different approach on locating virtual objects in the scene. They are placed relatively to identified reference objects in the image. First the virtual object is placed in the center of a taken image. Changing the size via a scrollbar is supposed to clarify the object's distance. ARCore is used only to render the 3D object. Afterwards, real reference objects in the room are identified via an object detection system. Image feature analysis then extracts image features for those reference objects. Those are then compared with image features found

by other devices. In case of correspondence, the reference objects' pose and position can be used to derive pose and position of virtual objects on all participating devices. The visual results do not look very convincing, though, which is why provided mapping procedures by ARCore and ARKit seem to be a better approach to rely on. The work claims to be working on Android smartphones and having no specific requirements for hardware resources. At least the concept itself may be implementable in a cross-platform compatible way, except rendering relying on ARCore.

A8: Secure Multi-User Content Sharing for Augmented Reality Applications [RKR19]

Ruth presents a design for an AR content sharing control module targeting security issues. The design is implemented for Microsoft HoloLens as a prototype named "ShareAR". The work brings up the case users within the same AR session distrust each other. Unwanted behavior by a user could be sharing inappropriate content or destroying other users' work on AR content. For this purpose, ShareAR brings in the idea of privacy policies for users and virtual objects. Users can obtain restricted abilities on transforming or even reading, that is visualizing other users' AR content. The works states a couple of pursued goals by the framework. These are divided in functionality goals to enable multi-user AR on the one hand, and non-functional security goals on the other.

- **Functionality Goals**

  Support physically-situated sharing: Co-located AR must support a consistent view with virtual objects coupled to the real world.

  Support physically-decoupled sharing: Remote AR must support showing virtual objects decoupled to only one specific location in the real world, as virtual objects need to appear in different rooms for different users.

  Support real-time sharing: User interactions and data manipulation must execute in real time for all participating devices.

- **Security Goals**

  Support granting/revoking per-user permissions: Users can allow or disallow others to join a common AR session.

  Support granting/revoking per-object permissions: Creators of virtual objects may keep the visualization private for the own device only, or make it public for all participants.

  Support physically intuitive access control: User-created virtual objects may not only have a single visualization for a participants. As an example, imagine a game card open for the owner, but covered for all other users.

  Support user control of incoming virtual content: To avoid spam, a user must be able to allow/disallow the visualization of virtual objects created by others.

  Support user control of owned physical space: Users may want parts of the room to be exclusively augmentable by themselves or allowed others.

As not all of the security goals are wanted in every multi-user AR scenario, they are not required to be implemented as fundamental part of the newly developed framework. Features accentuated to be not considered are user interface usability, network communication issues or localization of user positions.

A9: Just a Line[2]

Google provides the application "Just a Line" for promotion of ARCore and its features. The application enables the user to draw virtual white lines within an AR scene. The virtual elements constituting lines are placed floating in the room, close to the drawing device. Multiple users can join a session draw in collaboration and a consistent AR view. Connecting to each other is implemented per simple button touch. The application works without any explicit room scanning phase and can start drawing immediately. There are compatible versions for Android and iOS available. Just a Line is an impressing example of cross-platform multi-user AR, but available as a domain-specific single-purpose application only. Therefore, it cannot be used as foundation for further development. The application can be downloaded for free.

A10: The Machines

In return, Apple offers the maybe visually most impressive multi-user AR application. "The Machines" is a Multiplayer Online Battle Arena (MOBA) game, that can be placed on a real world tabletop. The application is was developed by the studio Directive Games, runs on iOS devices only and is chargeable. By moving nearer to the virtual objects, sounds become louder to the user [Cow17]. Multiplayer games can be played co-located with consistent view, as well as against remote opponents via internet.

A11: ShareAR: Communication-Efficient Multi-User Mobile Augmented Reality [RSGC19]

As described in Section 2.5, multi-user AR scenes can be synchronized with the help of cloud services like Google Cloud Anchors and Microsoft Spatial Anchors. An alternative synchronization concept is P2P communication, which is used by Apple ARKit. Ran et al. have developed ShareAR, a software component that is able to execute both synchronization concepts for the purpose of comparing them. The component can be integrated on top of existing multi-user AR components. This was tested with Google Cloud Anchors, Just a Line and AR Multiuser, an iOS software project provided by Apple. Concepts are compared in regard of three metrics.

- **Virtual Object Pose Accuracy:** A high accuracy of virtual objects on different devices means an accurate coordination of mapping data.
- **Virtual Object Pose Jitter and Drift:** In case of low accuracy of mapping data coordination, the coordination results can differ from frame to frame. The effect is virtual objects jumping from one pose to another from frame to frame.
- **End-to-end Latency:** Latency can be measured by the gap between the appearance time on the object-creating device and object-receiving devices.

A12: Second Surface [KHLI12]

Kasahara et al. present a multi-user AR system, that enables to draw virtual content within shared AR scene. To do so, virtual objects are oriented at detected at real world objects. Any participating device detecting the respective real world object views the related virtual objects. This is implemented via image based object recognition. Createable objects are drawings, texts and photos. The creation of drawings has similarities to *Just a Line* (**A9**), except one difference. While *Just a Line* allows to draw dynamically in three dimensions, users of this application draw their lines onto a two-dimensional canvas, which is placed into the three-dimensional AR scene afterwards. The canvas' position is located between the created device and the detected object. Furthermore, its rotation corresponds to the creating device.

[2] Just a Line - https://justaline.withgoogle.com

The system is based on a client-server architecture with handhelds as clients. Clients send data of generated contend and the device's pose to the server. In return, they receive data of other devices' generated content and respective pose data. Figure 3.4 depicts the architecture.

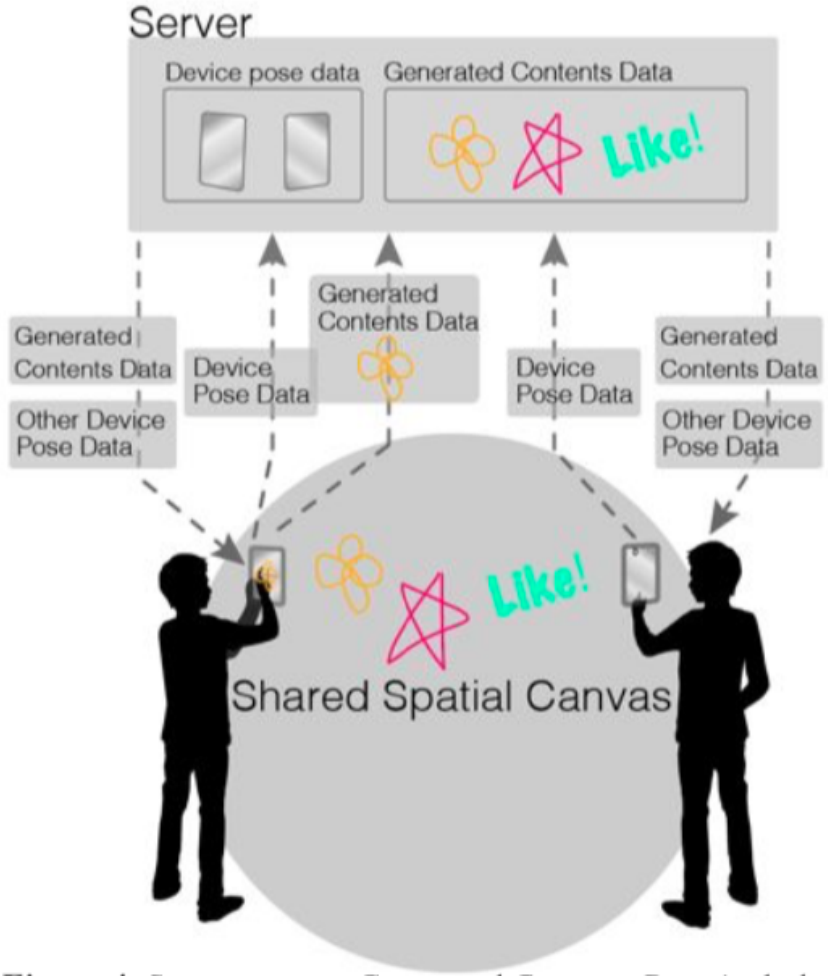

Figure 3.4: The client-server architecture of Second Surface - Source: Kasahara, 2012 [KHLI12] .

The presented system only supports handheld devices. The work does not mention, which devices exactly and which operating systems are supported. As there is no cross-platform compatibility addressed, it does not have all required characteristics. Furthermore, the system is not a reusable framework, but a domain-specific application.

A13: Face to Face Collaborative AR on Mobile Phones [HBO05] An older work from 2005 by Henrysson et al. presents a software architecture for marker-based multi-user AR. As virtual objects are oriented at detected markers, the AR view is consistent across participating devices. Supported devices are only feature phones with Symbian as operating system. An AR tennis game for two players was developed to evaluate the approach. Though the work is not based on up-to-date technologies anymore, it still provides valuable information due to its focus on the evaluation of user experience. A conducted study instructed participants in teams of two to play the tennis game against each other in three variants:

- **Face to Face AR:** Both players are facing each other and scanning a common marker

between them.

- **Face to Face non AR:** Both players are facing each other, but only playing a non-AR variant of the tennis game.
- **Non Face to Face AR:** Both players are separated from each other and each one is scanning his own respective marker. This results in a remote AR scenario.

Participants were afterwards asked about ease of collaboration. The results showed, that collaboration felt the easiest in the Face to Face AR scenario, while collaboration in the remote AR scenario felt the hardest. This can be interpreted as a indication that there are scenarios for co-located AR with consistent views, which may have advantages in terms of collaboration compared to remote AR scenarios or co-located AR scenarios with inconsistent views.

Neither does the presented architecture support modern-day handhelds or HMDs, nor does it support markerless AR. Therefore, it does not have all required characteristics.

A14: Ein Hologramm für alle (a hologram for all) [Pap20] Papenfußdescribes a very advanced approach to enable multi-user AR application. It uses anchor sharing with the help of the cloud service Azure Spatial Anchors by Microsoft. The recommended engine to develop the solution Unity. Devices considered to be supported are Android and iOS handhelds as well as the Microsoft HoloLens. The intended procedure to successfully share an anchor is described in four steps, as depicted in Figure 3.5. One device has to create an anchor attached to a point in the real world and save it on the Azure servers. Data to load the anchor from the server is transferred to all other participating devices. They can afterwards load the anchor data and find the anchor in the AR scene. All devices share the same anchor now.

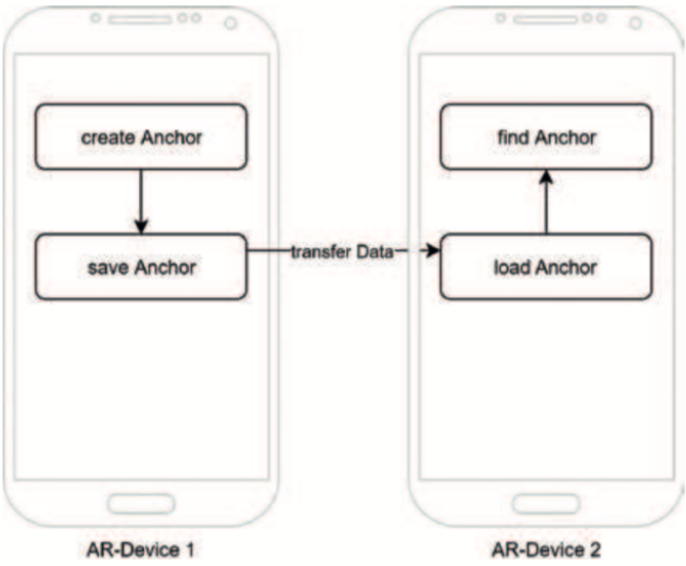

Figure 3.5: The steps of cloud-based anchor sharing - Source: Papenfuß, 2020 [Pap20]

The article addresses the necessity for user Interaction, but does not describe it in detail regarding different interaction modalities for various device types To communicate and coordinate interactions, the article recommends using the network component Photon with a client-server architecture and a dedicated server. The approach has huge similarities with this thesis' developed solution, which also uses Azure Spatial Anchors to share anchors. It differs in using a

network architecture with dedicated server, though, which was avoided in this work to rely less on extern services. All in all, the article presents a very sophisticated approach for modern-day cross-platform multi-user AR including implementable code snippets, but no ready-to-use framework.

3.3 Summary

Figure 3.6 shows a tabular overview over all examined related work in regard of meeting the stated requirement. As none of the approaches has met all eight requirements sufficiently, the desired solution constitutes a unique piece of software outstanding the related work. With **A1, A2, A6** and **A8** there are three frameworks in the list, though that aim for a similar idea, only with differing ranges of supported platforms and device types. This is why these three works are worth an extra look on their approaches to get inspiration for own design decisions. **A1** states challenges with high accordance to this work's own stated challenges. Especially the strong focus on coordinate management is an aspect, which has a high importance in this work, too. Managing coordinates is discussed in detail in Section 5.3. **A2** mentions the necessity to not only synchronize visuals, but also metadata and actions via network. The concept to enable this is described in the Sections 5.3 and 5.5. **A6** concerns the idea of using a server-client architecture with clients sending communication commands to the server. This is a concept also used in this work and described in Section 5.3. **A8** concerns possible restrictions on interaction with 3D content. Section 6.1.1 shows how interaction restrictions and permissions based on roles and states can be implemented with the developed framework. With research on related work completed, the conceptional design of the own approach can start.

	R1	R2	R3	R4	R5	R6	R7	R8
Requirement	**Multi-device**	**Cross-platform**	**Device types**	**Consistent AR view**	**Co-located AR**	**Markerless AR**	**Multi-user Interactivity**	**Framework**
A1: XD-AR	Yes	HoloLens, Tango, RoomAlive	Handheld, HMD, Projector	Yes	Yes	Yes	Yes	Yes
A2: Placenote	Yes	iOS	Handheld	Yes	Yes	Yes	Yes	Yes
A3: Collaborative Augmented Reality	Yes	Android, iOS	Handheld	Yes	Yes	No	Yes	Yes
A4: A collaborative Augmented Reality framework based on Distributed Visual SLAM	Yes	-	Desktop PC and Cameras	Yes	Yes	Yes	No	Yes
A5: Spatial	Yes	Yes	Handheld, HMD	Yes	Yes	Yes	Yes	No
A6: Multi-user interaction with 3D objects in Augmented Reality	Yes	iOS	Handheld	Yes	Yes	Yes	Yes	Yes
A7: PROMAR	Yes	Android	Handheld	Yes	Yes	Yes	No	Yes
A8: Secure Multi-User Content Sharing for Augmented Reality Applications	Yes	HoloLens	HMD	Yes	Yes	Yes	Yes	Yes
A9: Just a Line	Yes	Android, iOS	Handheld	Yes	Yes	Yes	Yes	No
A10: The Machines	Yes	iOS	Handheld	Yes	Yes	Yes	Yes	No
A11: ShareAR: Communication-Efficient Multi-User Mobile Augmented Reality	Yes	Android, iOS	Handheld	Yes	Yes	Yes	No	No
A12: Second Surface	Yes	Not mentioned	Handheld	Yes	Yes	Yes	Yes	No
A13: Face to Face Collaborative AR on Mobile Phones	Yes	Symbian	Feature phone	Yes	Yes	No	Yes	Yes
A14: Ein Hologramm für alle	Yes	Android, iOS, HoloLens	Handheld, HMD	Yes	Yes	Yes	Yes	No

Figure 3.6: Table: Related work overview. Red: Requirement is met insufficiently

4 Conceptual Solution

Now that research on related work is discussed, this chapter will take a look on the conceptual design of the developed framework „XshARe“. This framework will be used to develop multi-user AR applications. Section 4.1 describes the software project's architecture, considering interfaces to the outside, as well as the internal component structure and control flow between internal components. This is followed by Section 4.2 taking a look on determined generic use cases concerning the AR applications. The product functions derived from those use cases are supposed to be provided by the framework. In Section 4.3 , the product functions get set into relation of the stated requirements. The conclusion will show, that all product functions are necessary parts of a solution to meet the stated requirements considering used technologies.

4.1 Architecture Design

The developed framework solution has to interact with a couple of external components and actors. As developed multi-user AR applications are supposed to use the framework as foundation, the solution requires an easy-to-use application programming interface (API) for developers to utilize the framework and its provided services for their purposes. In the same way the solution has to interact with its own foundation, example software project for sharing anchors. User inputs from various devices require to be read out and interpreted to commands. This includes onscreen button navigation as well as interaction with virtual objects within the AR scene. Network communication is implemented by using a third-party network component. Therefore, the solution needs a concept to interact with the component's networking interface. Finally, the solution requires to read and modify 3D model data from the active Unity scene and attached metadata. Figure 4.1 shows all mentioned elements interacting with the solution.

To manage interactions with external components, the solution has a modular internal structure. It consists of five exchangeable components using defined interfaces to communicate with each other. This way, single aspects of the implementation can be edited or exchanged within the respective component and without considering other components and their concerning aspects. Figure 4.2 shows the solution's internal component architecture.

- **SpatialAnchorsCoordinator:** This component encapsulates all extensions of the integrated example software project for sharing anchors. This includes extending the button menu for executing anchor sharing with an additional user interface for network connection. Furthermore, a room number is read out, shared and used as ID for anchor locating. Product functions **P1** to **P4** are implemented within this component, all executed via

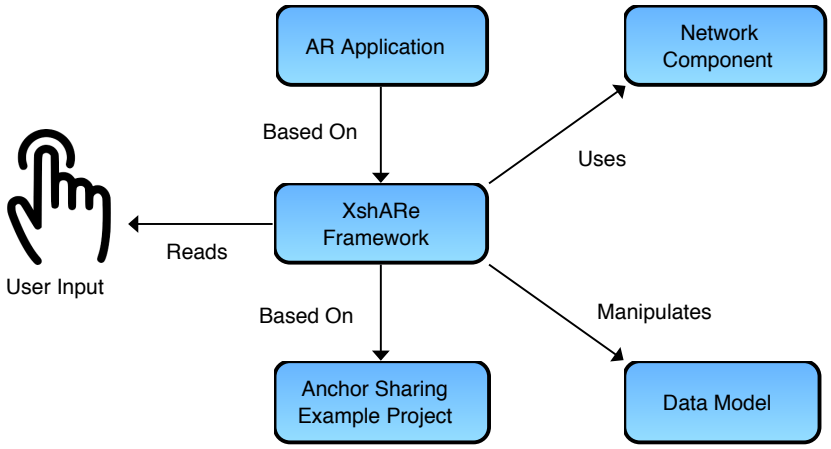

Figure 4.1: Extern components and actors interacting with the framework.

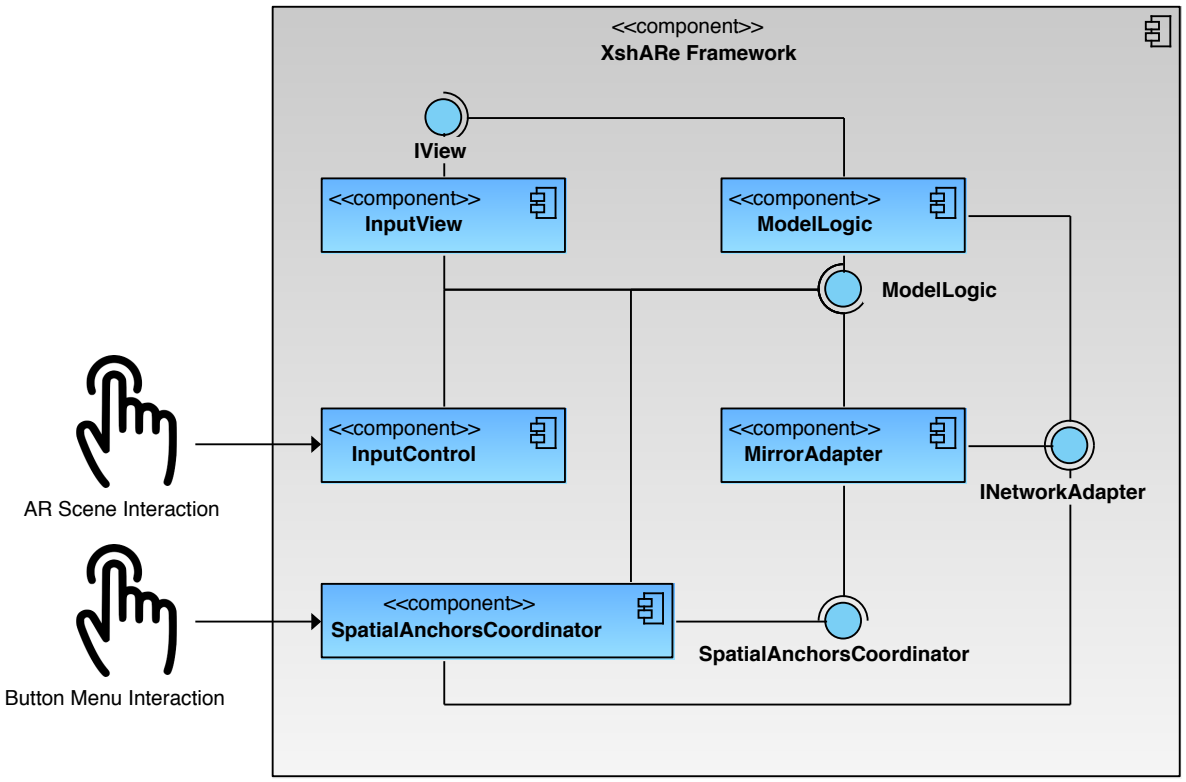

Figure 4.2: The frameworks's internal component architecture.

buttons on the screen overlay in front of the AR scene. The component's jobs require access to components for network communication and model data. Section 5.2 goes into further detail.

- **MirrorAdapter:** The MirrorAdapter is responsible for interacting with the third-party network component. This includes registering as host, searching for an available host or server, joining as client and sending commands for data modification to the server. Data modification commands include moving and rotating to specific coordinates. As each connected application holds GameObjects within its own differing coordinate system, the component also executes a concept to calculate global coordinates to local coordinates relative

to the anchor and vice versa. The component implements the interface INetworkAdapter. If a developer decides to exchange Mirror with another network component, he needs to develop a new adapter component implementing the interface. For cases of calling business logic executed on server-side only, MirrorAdapter requires to access the respective business logic component. Section 5.3 takes a look on the component's internals.

- **InputControl:** This component reads out user input and interprets them to interaction commands to manipulate 3d model data. If an interaction command is detected, the command is called on local the local business logic component to validate the modification request. Section 5.4 discusses specific input modalities for supported device types.

- **ModelLogic:** This is where business logic is calculated. Each connected application is running a local variant of this component. This is necessary, because there is eventual program logic, which is related to data held locally, like selection information. The component SpatialAnchorsCoordinator sends the order to officially start the AR session. Data modification requests come come from the InputControl component. Then requests are then validated. Depending on the validation result, model data is eventually modified. This modification has to be synchronized via Mirror, which is supported by the MirrorAdapter component. In case of eventual server-side business logic required, server requests have to be sent through the MirrorAdapter, too. In contrast to the other components, this one has to be overridden for each domain-specific application variant. This is done by implementing a class inheriting form the ModelLogic class and partially overriding its protected methods. It is the central interface for the developer to se the framework for his own specific purposes. Section 5.5 will explain this in further detail.

- **InputView:** The ModelLogic component provides components implementing the IView interface to register for notification. Multiple view components can be notified in case of changed model data. The only IView component considered yet is the InputView. This component extends the AR view by visualizing user input and selection. Section 5.4 describes the InputView's visualization and the relation to InputControl and ModelLogic.

To get an understanding interaction between components, Figure 4.3 shows the sequence of an exemplary case. Depicted is the control flow triggered by the command for creating a new virtual object within the AR scene. In case of a handheld device, the user has to execute a double tap onto the screen. The InputControl component reads user input at each frame and interprets it as assumed interaction requests. In this case, the request is the method AddGameObject on the ModelLogic component. A domain-specific application can override this method to implement an own request validation. When object creation at the requested position is confirmed, a new GameObject can be created. To be synchronized on all connected devices, the creation has to be executed on server-side. This is triggered by calling the method CreateSyncGameObject. Every newly created object is automatically selected on the interacting device. This has implications for the InputView, also visualizing the selection. That is why InputView has to be notified. The component then requests information about the currently selected GambeObject and its pose. Selection visualization can finally be updated.

4.2 Product Functions

Using AR applications with the presented *XshARe* framework is considered to include a workflow of three generic phases, as shown in Figure 4.4. In the first phase, named as *Establish network connection*, the user is establishing the network connection with other participating devices. Te

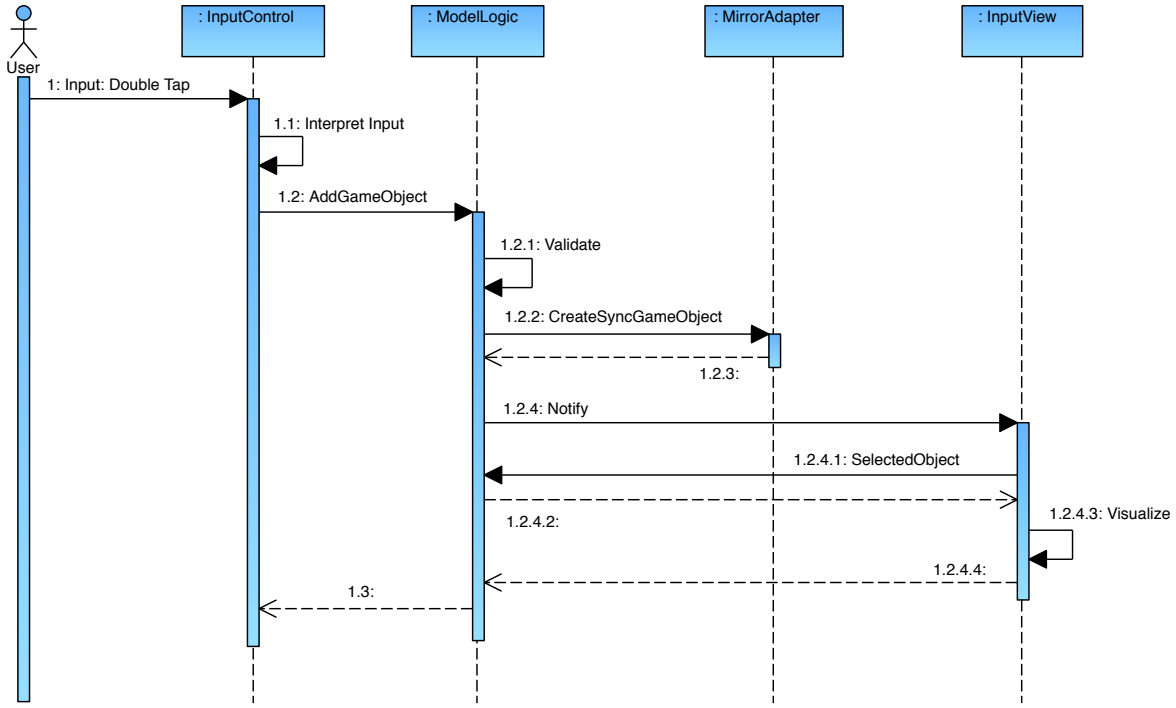

Figure 4.3: Example sequence diagram: Creating a new GameObject within the multi-user AR scene.

second phase, named as *Establish world anchor*, is concerned with setting, sharing and location a common anchor. These two first phases constitute the preparation of the multi-user AR session, which is run in the third phase, named as *Runn AR session*. While phase 1 and 2 can be reused for different multi-user AR applications, the third phase is varying. Figure 4.4 shows an overview over all three phases in a state chart. Each phase comes with its own product functions, that are discussed in the following.

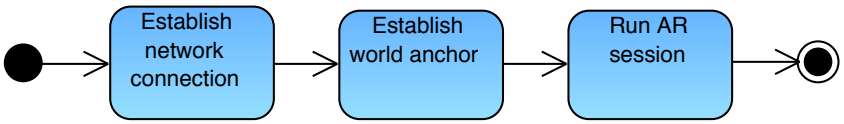

Figure 4.4: The three generic user workflow phases in every XshARe application.

To enable multi-device interaction between multiple devices, network communication between those is required. The solution is designed to enable network connection with a server-client architecture. Though the intended network architecture does no require a dedicated server, it still requires a connected device acting as host, which means acting as server for the session and being a client like all other devices at the same time. Users starting a multi-user application need to decide between two actions to establish a connection with each other. Figure 4.5 depicts the relations between network-related product functions.

- **P1: Start as Host**
  Precondition to start a network connection a as host is having the application started and no other device already being registered as server. The user is supposed to tap a button on the screen overlay in front of the AR scene to easily start as host. The application will also register itself as the first connected client. Postcondition is a participating device

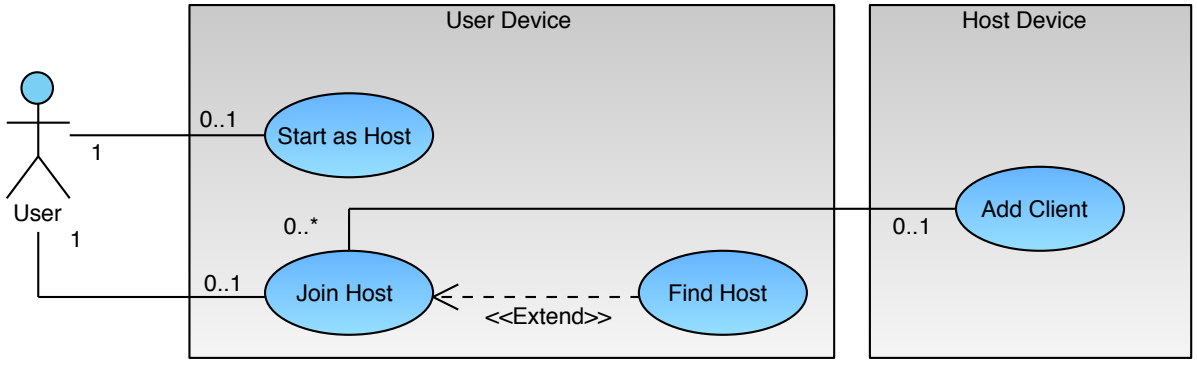

Figure 4.5: User interactions for network establishment.

acting as server with itself as first connected client. The device's GUI switches to the next button menu concerning shared anchors.

- **P2: Join Host**
  Instead of starting as host, the user can also choose to join a running server as client. Precondition in this case is a single application already running and using Mirror networking connected to the same WIFI network. The joining action is extended by a command to find an accessible host. After the accessible host is found automatically, the join can be confirmed. Both functions are triggered by button tab on the screen overlay. Postcondition is an additional device registered as client by the host. The device's GUI switches to the next button menu concerning shared anchors, just like after host establishment.

While 3D model and metadata are synchronized via the network component, the anchor is shared via a cloud service. There are example software projects available, already running an anchor synchronization within an AR scene. To succeed the synchronization process, users must go through a couple of button menus. Section 5.2 will go into detail on the menu navigation. Essential product functions within these button menus are creating and locating a shared anchor. Figure 4.6 shows the two product functions within a use case diagram.

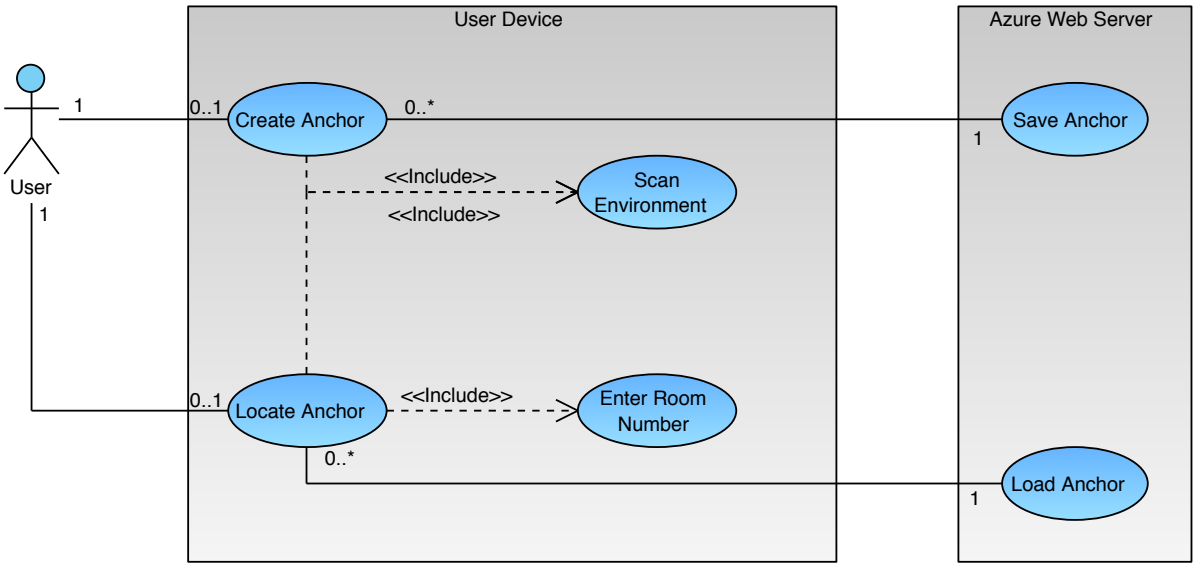

Figure 4.6: Required interactions to share anchors.

Synchronized locating and mapping data are stored on the cloud for a short time. Each session's data gets a data entry with a unique ID, the room number. In anchor sharing software examples this room number has to be read out by the anchor creating user and put into a text field on the screen overlay by each user to locate the anchor. The newly developed framework is supposed to use the network connection to automatize room number sharing. Figure 4.7 depicts the required changes.

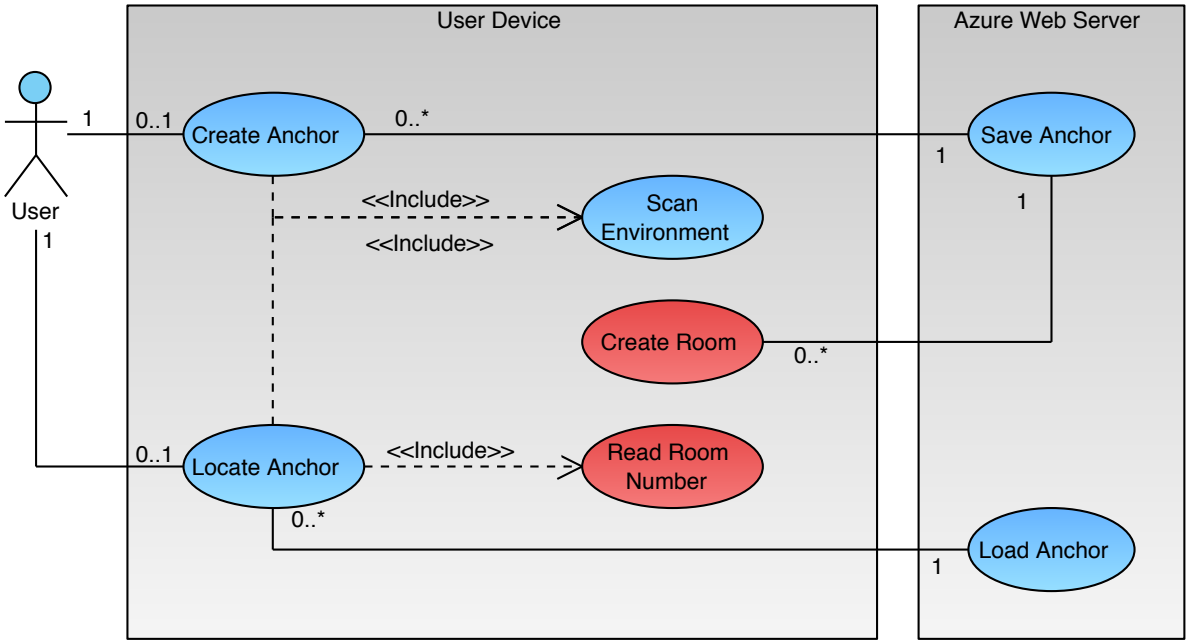

Figure 4.7: Use case diagram: modified shared anchor interactions.

- **P3: Create Anchor**
  Precondition to create and share a new anchor is a device connected to a local network server. This can also include the host itself. The anchor setting process is started by tapping onto a button. Following steps include scanning the environment for horizontal surfaces and setting a location for the anchor on these found surfaces. Afterwards, locating and mapping data is uploaded to the cloud, which return the unique room number. The number is shared with all participating connected devices, also with those, which connect after setting the anchor. As a postcondition, the user returns to the beginning button menu, with which he can now go on to locating the set anchor.

- **P4: Locate Anchor**
  To locate an anchor, precondition is to have anchor data accessible on the cloud and a room number shared with the user's device. Locating an anchor requires scanning the environment again. After comparison of data between the scanning device and and the cloud service, considering the respective room number, the device gets the exact anchor position back from the server. Postcondition is a running AR scene with a synchronized anchor, at which virtual objects can be oriented at. The generic preparation phase is over and the specific part of the AR application can start.

After the preparation is done, the actual multi-user AR session can begin, including virtual objects placed into the real world. Users are supposed to be able to interact with those virtual objects. To do so, they need input modalities depending on the used device type. Handhelds,

for example, are predestined for input via their touch screen. The presented XshARe framework provides six different kinds of generic interaction commands with virtual objects. They all share a running multi-user AR session as precondition as well as on postcondition. Several input variants have further specific pre- and postconditions, though. These are discussed below. As the applications require to keep the 3D model synchronized and a strictly server-authoritative network component is used to implement network synchronization, each local command for data modification requires a modification command called on the host or server. Figure 4.8 gives an overview over all generically provided input variants.

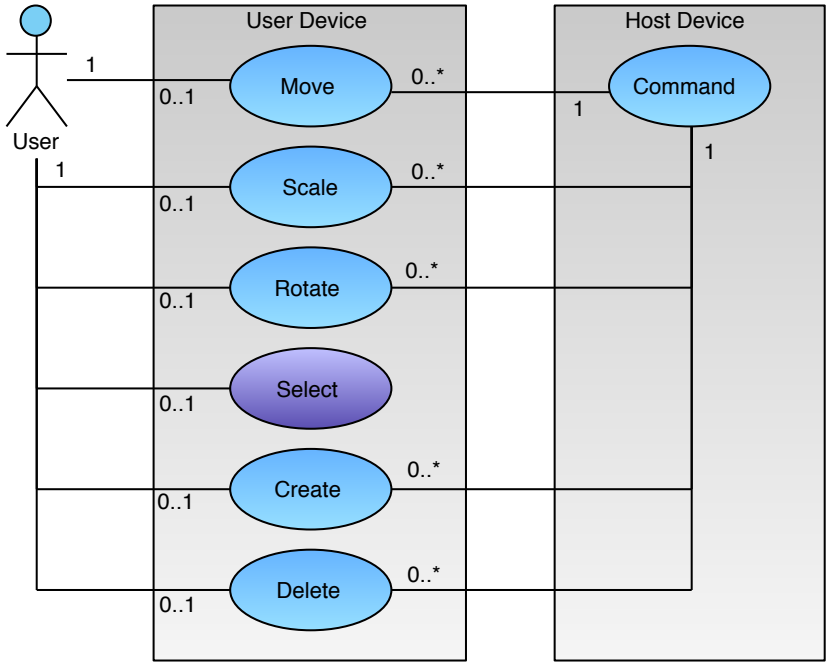

Figure 4.8: Possible interactions with 3D model.

- **P5: Create virtual object**
  The user places a new virtual object within the AR scene at a specific position. There are no additional preconditions. As a postcondition, the AR scene consists of an additional virtual object synchronized on all participating devices, including possible attached metadata.

- **P6: Select/deselect virtual object**
  The user selects or deselects a selectable virtual object within the AR scene. Precondition is at least one selectable virtual object included in the AR scene. Further interaction can later be executed for data modification on the selected virtual object. Each user has his own local selection information, which is not needed to be held synchronized via all devices. This is why the selection does not lead to a command on the server.

- **P7: Delete virtual object**
  The user removes a selected virtual object from the AR scene. As precondition, there has to be a virtual object included in the AR scene, that is locally selected by the interacting user. The virtual object and related metadata needs to be removed on all connected devices as postcondition.

- **P8: Move virtual object**
  The user moves the selected virtual object to a new specific position within the AR scene.

Precondition is an existing selected virtual object again. After the interaction, the virtual object is supposed to be located at the desired destination, visually synchronized on all devices.

- **P9: Scale virtual object**
  The user makes the selected virtual object bigger or smaller. Scaling is supposed to be handled evenly in all three dimensions by default. Precondition and postcondition are according to **P8**.

- **P10: Rotate virtual object**
  The user rotates the selected virtual object in one of two directions. Rotation is only provided around the z-axis, as this variant makes the most sense in considered scenarios. Many real world objects, that are placeable in a room, have an identifiable up side and a down side and are mostly placed upright. Take a piece of furniture, like a chair, as an example. A virtual chair in an AR app will probably rarely have to be tilted, as well. But in a furniture placing app, rotating the chair around the z-axis is required a lot. Precondition and postcondition are according to **P8**.

4.3 Context between Product Functions and Requirements

The discussed product functions are derived from the stated requirements and architectural boundaries from used technologies. This section recalls the requirements stated in Section 3.1 and discusses the relation to discussed product functions.

- **R1: Supporting multi-device sessions**
  Each participating user has his own device with an own display as well as camera. This requires a running network connection between those devices. Computer networks work according to the principles of basic network architectures, which describe the interaction between individual connected participants. The developed solution is supposed to use a client-server architecture for network communication. One participating device during a multi-device session has the role of the host, acting both as server and client. **P1** describes the product function of starting the user as host, while **P2** describes joining as client only. The client-server-based connection is established by clients finding an available host and getting registered as clients on the host device.

- **R2: Supported cross-platform range**
  The supported cross-platform range is modeled as a set of systems able to run AR applications. This requirement is not connected to any specific product function, as it mainly relates to the made choices of used technology for development.

- **R3: Supported device types**
  Interactions, especially those in **P5** to **P10** have to be implemented differently for each supported device type. This mainly refers to the considered device types handheld and HMD. While handheld devices need interaction modalities based on touchscreen input, HMDs provide a wider and more complicated range of input types. Section 5.4 goes into further detail on interaction modalities.

- **R4: Consistent AR view**
  Holding the AR view consistent between devices is one of the core features of this work. The desired approach to enable this is to share anchors via an external cloud service. Anchors in AR scenes are reference points strictly attached to a real world point and

providing 3D coordinates to orientate virtual objects. **P3** is concerned with creating an anchor and saving it in the cloud. **P4** describes loading and locating shared anchors.

- **R5: Co-located AR**
  The required Co-located AR approach has two implications. Firstly, a consistent AR view does not only require virtual objects consistently placed in relation to each other, but also in relation to the real world. Secondly, participating users can possibly all use the same local computer network for communication, while remote AR approaches will most likely have to make use of communication via internet only.

- **R6: Markerless AR**
  Instead of using detectable markers for orientating anchors, anchor poses are located from room scanning data. Both **P3** and **P4** include room scanning as a function to enable the anchor sharing process. Saving an anchor in the cloud requires to collect mapping data during the AR session to upload. To locate a shared anchor, the uploaded mapping data needs to be compared with the mapping data on the local device.

- **R7: Interactivity for multiple users**
  The product functions **P5** to **P10** are intended to provide interactivity with the AR scene for each user independently from each other. Intended interactions include creating and deleting virtual objects in the AR scene as well as manipulating their data for positioning, rotation and scale.

- **R8: Framework character**
  All presented product functions refer to the end user in interaction with developed AR applications. They are all provided for every application build upon the framework and can be extended by the developer for each domain-specific application, if necessary.

5 Implementation

As presented in Chapter 4, the developed solution is a framework for Unity structured in several components. After discussing structure and workflow on an abstract level, this chapter goes further into implementation details. Section 5.1 takes an overall look on the Unity project as a whole. The following chapters discuss design and implementation decisions within the encapsulated single components. Section 5.2 begins with the SpatialAnchorsCoordinator, a component concerned with the Integration of the Spatial Anchors Examples project used as foundation. The integration of the third-party network component Mirror for communication via the component MirrorAdapter is then presented in Section 5.3. To handle user interactions, the components InputControl and InputView are used. Those are discussed in Section 5.4. User interactions have an impact on business logic and data manipulation, which is managed by the component ModelLogic. Section 5.5 is concerned with its details.

Like every Unity project, the developed solution is consisting of a couple of assets combined within scenes in a hierarchical structure. Assets include resources like 3D models and images, but also code scripts to execute program logic dynamically during runtime. All implemented scripts were written in C# during this work.

Figure 5.1 shows the component's main classes and their relation relation to each other.

5.1 The Unity project

The provided Azure Spatial Anchors Examples project comes with four AR scenes, each with a different functional scope.

- **AzureSpatialAnchorsBasicDemo:** The user can place an anchor within an AR scene. A GameObject is instantiated with the placed anchor's position and rotation. This is a case of single-user AR.
- **AzureSpatialAnchorsCoarseRelocDemo:** The user can relocate his position. This is a case of location-based AR.
- **AzureSpatialAnchorsNearbyDemo:** The user can place multiple anchors to build a signage system for wayfinding. This is also a case of single-user AR.

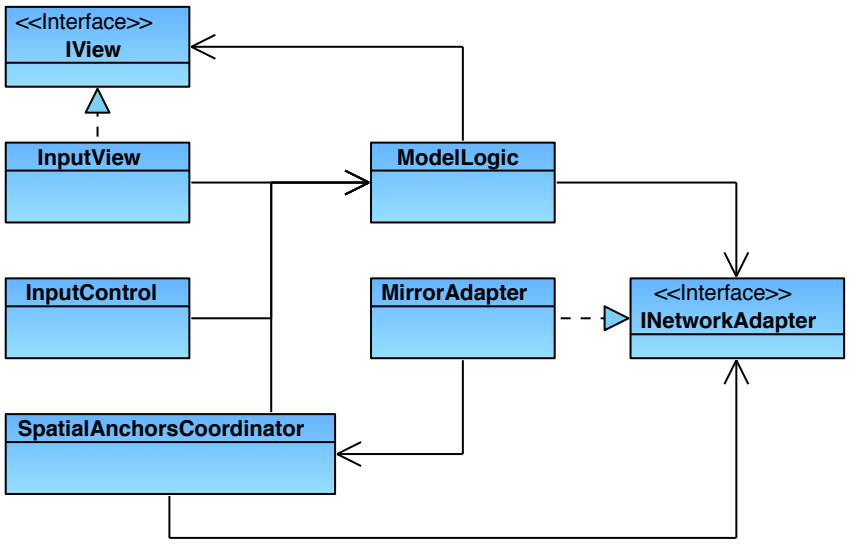

Figure 5.1: The relation between main each component's main class.

- **AzureSpatialAnchorsSharedAnchorDemo:** The user can place an anchor within an AR scene and be uploaded to a Microsoft Azure web server. In a second step, uploaded anchors can be located. This is a case of multi-user AR and therefore the scene of choice to build up on as foundation.

In addition, the project provides a starting scene managing a selection menu. Multiple scenes can be compiled to a single application, which can jump from scene to scene. To be considered by the selection menu scene, additional scene names do not only have to be added in the Unity build settings, but also need to be named starting with *AzureSpatialAnchors*.

To develop on top, the scene *AzureSpatialAnchorsSharedAnchorDemo* was duplicated. To constitute the new framework, the scene is modified and extended. This extended scene can already run as a standalone-application with very rudimentary and reusable features. This scene *AzureSpatialAnchorsXshARe* can again be duplicated and modified to develop an individual, domain-specific multi-user AR application. Figure 5.2 shows a screenshot of the AR scene for the mill example, including all GameObjects statically instantiated at the scene start. Those GameObjects are always the same and no further GameObjects are supposed to be added or removed within the editor. All further GameObjects needed for application-specific need to be instantiated dynamically at runtime via scripting.

- **Directional light:** This is a standard GameObject existing in each newly created Unity scene. It holds components concerning the scene lighting. Like any GameObject, the Directional light can be placed and rotated individually. Lighting was none of this work's concerns.
- **CameraParent:** The Azure Spatial Anchors examples come with this GameObject concerned with choosing a suitable camera depending on the used device type, that is either handheld or HMD (HoloLens). This is required to provide cross-platform compatibility.
- **AzureSpatialAnchors:** A GameObject provided by the AzureSpatialAnchorsSharedAnchorDemo scene is AzureSpatialAnchors. It holds MonoBehaviours used to handle and

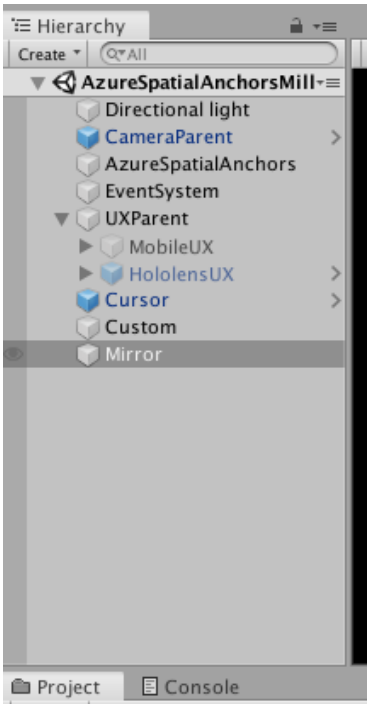

Figure 5.2: The common scene structure of a multi-user AR scene working with XshARe, depicted at the mill game example.

run the scene's workflow logic. This is done especially by AzureSpatialAnchorsSharedAnchorDemoScript. This MonoBehaviour is base class and replaced by the self-developed SpatialAnchorsCoordinator. Section 5.2 goes into further detail on this class.

- **EventSystem:** The EventSystem is a standard GameObject also existing in each newly created Unity scene. It is supposed to be used to handle interaction with the 3D model and even provide object selection. As the solution requires integration of different input types, like touch and mouse click, the EventSystem is ignored. Its functionality is replaced by the component InputControl, further discussed in Section 5.4.

- **UXParent:** While AzureSpatialAnchors holds program logic, all different buttons to interact with either handhelds or HoloLens are stored within a hierarchic structure starting with the UXParent. The structure is edited to provide new buttons for the extension to provide Mirror network connection.

- **Cursor:** The Microsoft HoloLens requires a cursor to interact with the scene. This GameObject also provided by the The Azure Spatial Anchors examples holds logic to handle the cursor.

- **Custom:** This is the main GameObject, that should be of interest for developers using the framework. It is supposed to hold all business logic within the MonoBehaviour *ModelLogic*, which is to be inherited and replaced by a new business logic class. It is recommended to add an additional MonoBehavour as configuration file to provide access to all needed GameObject prefabs and additional preferences.

- **Mirror:** To use Mirror within a scene, it is required to create a GameObject holding a NetworkManager component. This component is where networking preferences can be

edited. The GameObject holds the self-developed MirrorAdapter in addition. See Section 5.3 for further details.

In addition to an AR scene running on end-devices, it is advisable to use a second test scene for testing non-AR aspects of the application on the desktop. A respective test scene is also provided part of the developed solution. It lacks all Spatial Anchors elements like AzureSpatialAnchors, UXParent and Cursor. Desktop testing is highly recommended for faster debugging. While interactions are executed via finger touches on handhelds, these are simulated via mouse click by the self-developed helper class TouchMouseSimulator.

Unity provides the possibility to have fields in MonoBehaviour scripts shown and editable within the Unity editor. In the script itself, these fields can be set on null or default, while they start runtime with the values put in via editor. While desktop testing, their value can also be shown in real time in the editor. To do so, the header attribute [SerializeField] hast to be written above the respective field, like shown below:

```
[SerializeField]
[Tooltip("Prefab for Azure Spatial Anchors Room Information.")]
private GameObject roomPrefab = null;
```

This is the only method in Unity to load prefabs of GameObjects from the assets folder into the running scene, which will be a topic in several of the following chapters.

5.2 SpatialAnchorsCoordinator: Integration of the Spatial Anchors Examples project

As already mentioned, Unity provides building up hierarchical component structures. Each component can hold multiple additional components as children. These children as well as a possible parent are accessible via the Transform object, each component owns. In addition to that, the Transform object also holds all information about position, rotation and scale of an object. Considering a GameObject as a component example, these informations would be used for rendering the represented virtual object in the 3D scene. Another special kind of component is the MonoBehaviour, the base class for each Unity script to inherit from. This is also true for the four AR scenes of the Azure Spatial Anchors Examples. Although they have different functional scopes, common features are inherited from the base classes InputInteractionBase and DemoScriptBase. All classes inheriting from from DemoScriptBase spawn a GameObject at an anchor's position, whenever it is created and placed. Originally, this is a small cube. The last set shared anchor is always represented by an instance of the class CloudSpatialAnchor, which provides being shared via internet. Figure 5.3 gives in overview over the class structure. The self-developed class SpatialAnchorsCoordinator inherits from the AzureSpatialAnchorsSharedAnchorDemoScript. The cube to be spawned at the shared anchor's position is replaced by an image of an anchor.

The AzureSpatialAnchorsSharedAnchorDemo scene provides workflows to create and share anchors as well as to locate them during an AR session. In case the user wants to create a new anchor, he has to place it via raycast on a real world surface found by surface detection. The user can reset the anchor position as often as he wants before confirmation. The rotation is respective to the rotation, that is considered as the coordinates (0,0,0) in the AR scene. The coordinate system is applied to the real world in the first frames of environment scanning. After the anchor is confirmed, the application requires further mapping data and asks the user to

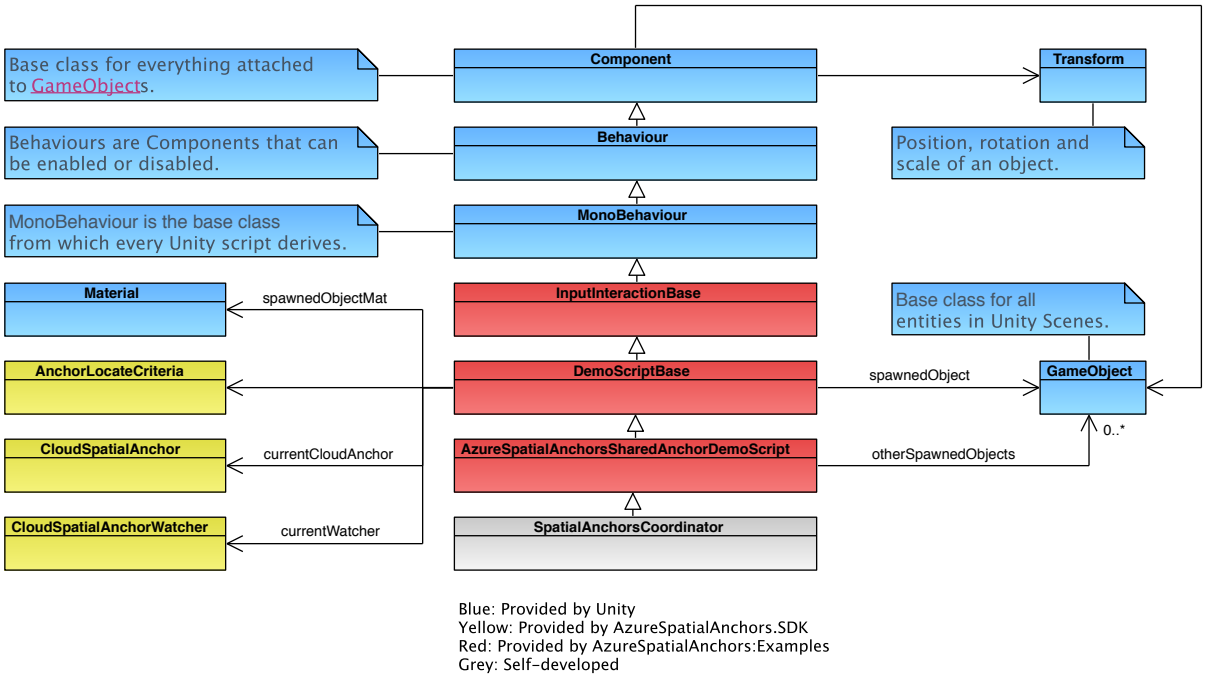

Figure 5.3: The class structure of AzureSpatialAnchorsSharedAnchorDemoScript and SpatialAnchorsCoordinator.

scan the environment until it is mapped sufficiently. The data can be uploaded to the Microsoft Azure web service after the scanning process is finished. The client gets the unique room number in return to be able to locate the respective anchor again. The user gets back to the start menu and is able to either create another anchor or to find an anchor. In the original AzureSpatialAnchorsSharedAnchorDemo, the room number has to be typed in manually to do so. When the anchor localization finished successfully, the man session can start. Originally, this is a simple cube placed at the anchor. The developed XshARe framework provides an interface to extend this session with individual data and logic (see Section 5.5). Figure 5.4 gives an overview over the workflow during the anchor preparation process.

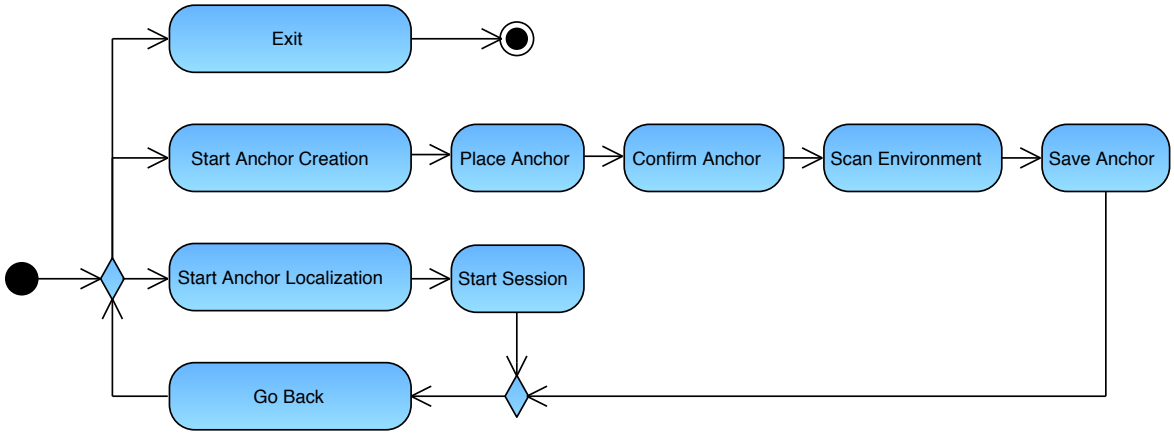

Figure 5.4: Workflow during the anchor preparation process of AzureSpatialAnchorsSharedAnchorDemoScript.

All steps during this workflow are started by pressing buttons. The *AzureSpatialAnchorsSharedAnchorDemoScript* provides an enumeration for the state, the scene currently is in. By navigating through the button menus, the so-called field *AppState* can switch over to the next value. Figure 5.5 depicts the possible routes to get from one state to another. The SpatialAnchorsCoordinator edits and extends the original workflow to fulfill all requirements. Instead of directly starting with the flow choice between create flow and locate flow, the button menu for establishing the network connection is implemented upstream. This is done by manipulating the user interface when the AppState is initially set on *Choose Flow*. The last state within the *Create Flow* super-state, which is implemented as a second field, is *Stop Session*. As this is the first state after *Saving CloudAnchor*, this is also the first state having access to a new room number. Originally, this room number is shown in the AR scene next to the spawned GameObject. Users are supposed to read it out and type it in later for locating anchors. The SpatialAnchorsCoordinator, however, gives the MirrorAdapter the order to spawn a new synchronized GameObject holding the room number as metadata. Section 5.3 explains the functionality of the component MirrorAdapter, while metadata handling is further explained in Section 5.5. When room numbers are automatically synchronized, of course there is no need to provide a text input field for it any longer during the *Locate Flow*. This is why the SpatialAnchorsCoordinator manipulates the original workflow, so that the AppState *Input Anchor Number* responsible for the room number input is directly skipped. The last state within the super-state *Locate Flow* is *Stop Session For Query*, in which originally a single GameObject is spawned at the anchor. when this state is reached, the SpatialAnchorsCoordinator notifies the component ModelLogic, that the interactive AR session can finally start, as the device has found the shared anchor to orientate GameObjects at.

Figure 5.6 shows a screenshot of a compiled example application. The orange surface with with black borders are a user feedback on surface detection results. Detected surfaces in this case are the top of the table and parts of the floor. Floating in front of the AR scene is the button menu for choosing the network connection at the beginning. While the button visualization is executed by the SpatialAnchorsCoordinator, pressing them triggers methods in the MirrorAdapter, as they are concerned with Mirror component interaction.

5.3 MirrorAdapter: Integration of Mirror Networking

The MirrorAdapter component has the task to handle network communication with the help of the third-party network component Mirror. Two classes are part of the component: MirrorAdapter and MirrorServerBehaviour. The MirrorAdapter class implements the interface INetworkAdapter and is accessible by other components via this interface. The idea behind encapsulating program logic concerning network communication in an own component is to be theoretically able to exchange used network-components easily. The implementation process showed, though, that commitment on a third-party network component has a huge and inevitable impact on the overall project structure. So does Mirror dictate multiple modalities for the implementation:

- GameObjects can only be synchronized on server-side.
- Data modification can only be synchronized on server-side.
- Methods on server-side have to be called via a special [Command] header.
- Only primitives can be sent as parameters for commands.

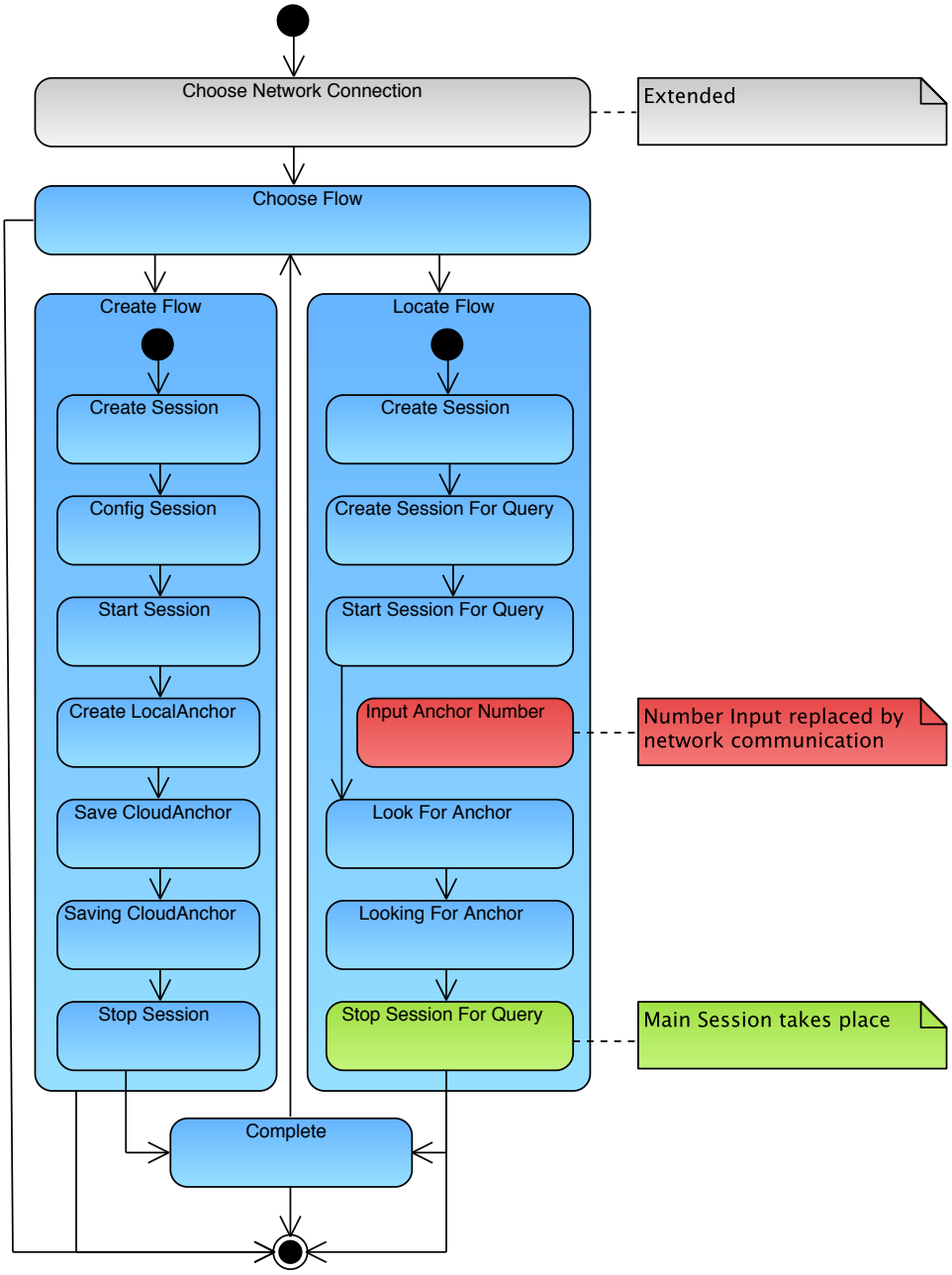

Figure 5.5: Example sequence diagram: Creating a new GameObject within the multi-user AR scene.

- As GameObjects are no primitives and have to be synchronized on server-side, they also have to be instantiated and destroyed on server-side.
- Server commands have no return value.
- Server commands have to be implemented within MonoBehaviours.
- All prefabs considered for synchronized GameObjects have to be registered in an array held by Mirror's NetworkManager component.

The following code snippet shows the header of the server command to create and synchronize

Figure 5.6: Screenshot of the user interface including the button menu for network connection.

a new GameObject:

```
[Command(ignoreAuthority = true)]
 public void CmdCreateObject(int prefabIndex,
 Vector3 localPosition,
 Quaternion rotation,
 int randomID)
  {
  ...
  }
```

The *prefabIndex* concerns prefab used to initialize the GameObject with. It is the index of the prefab in the Mirror registration array for prefabs. *LocalPosition* holds coordinates to locate the position in the AR scene and rotation coordinates for the respective rotation. As the command cannot communicate pointers, a workaround is necessary to access the lately created GameObject on client side. For this purpose, the commanding client generates the unique ID *randomID* and transfers it as parameter. After synchronization, the client can then find the lately created GameObject, for example to command further data manipulation on it. All server commands from the MirrorAdapter are implemented solely inside the class MirrorServerBehaviour, which is supposed to be accessed only by the MirrorAdapter. Figure 5.7 shows the component's class structure.

Not only does the MirrorAdapter have the task to execute data manipulation tasks. First, it has to provide establishing the network connection. Pressing the buttons in the network connection menu triggers methods implemented within the MirrorAdapter. If the user presses to connect as host, host establishment is called on the NetworkManager of Mirror, followed

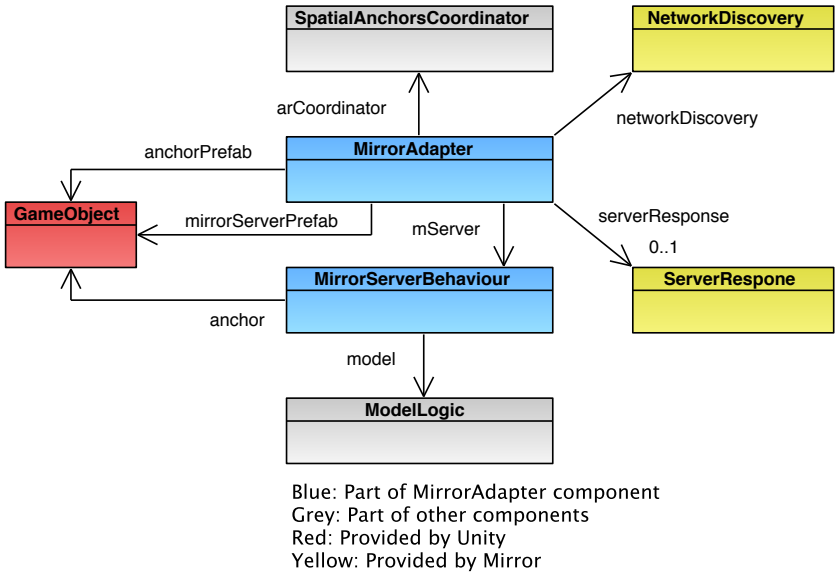

Figure 5.7: The class structure of the MirrorAdapter component.

by registration at a NetworkDiscovery object. The NetworkDiscorvery is a Mirror component, that automatically finds registered servers on the network. This prevents the developed solution to have the requirement to put in the a host IP address manually. A host device can of course instantiate and synchronize GameObjects without a dedicated server command. The host uses this to create and share a GameObject with a MirrorServerBehaviour component. Once spawned on all connected devices, each application's MirrorAdapter can access the synchronized MirrorServerBehaviour to call server commands on it. In case of choosing the client option the NetworkDiscovery searches for available servers and returns a ServerResponse object in case of success. After finishing network connection.

The MirrorServerBehaviour also provides a command to call any business logic method on server-side only. Implementing as much business logic as possible on server-side is of cause advisable anyway, but not necessarily required by the developed framework.

A main challenge of network synchronization in multi-user AR applications is traversing coordinates of position and rotation. Figure 5.8 shows a multi-user AR scenario from above. Two users are viewing a virtual object at the same shared anchor located in the room. As they start from different positions, the coordinate systems laid over the real world have zero points at different positions and are rotated differently, as well. Therefore, the viewed virtual object (GameObject in Unity) has different global coordinates on each device to be visualized at the same real world position.

But how can GameObject poses be synchronized, when their global coordinates differ in a consistent multi-user AR session? The easiest way would be to create, share and locate a new anchor for each new GameObject. The GameObject could then be placed directly at the position and rotation of the shared anchor. This approach would not provide pose manipulation in real time, though. Furthermore, it would require much more anchors and therefore produce much more data to save and process. To orientate all GameObjects at a single anchor, all coordinates have to be converted to a coordinate system with its zero point at the anchor. These coordinates then have to be synchronized. Figure 5.9 depicts this approach.

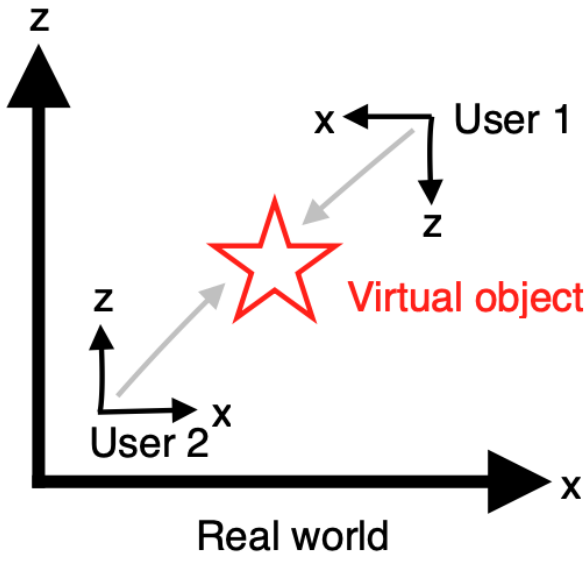

Figure 5.8: Scenario in top down perspective: two devices visualize an AR scene with different coordinate systems.

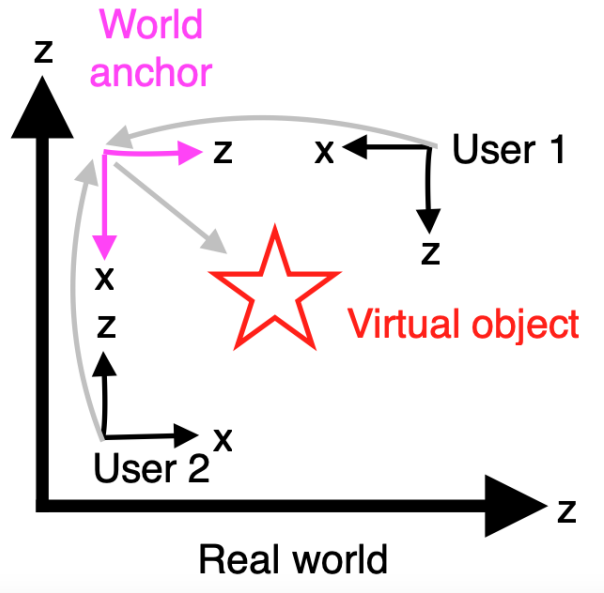

Figure 5.9: Scenario in top down perspective: Both devices use the shared anchor pose as coordinate common system.

This approach is implemented by placing a synchronized empty GameObject *anchor* directly oriented at the shared anchor. The SpatialAnchorCoordinator orientates it at CloudNativeAnchor object *currentCloudAnchor*, which holds coordinates for the set point in the real world. The global coordinates of the anchor GameObject different on each device, as the GameObject only synchronizes coordinates, if it holds a NetworkTransform component. All additional spawned GameObjects can now be attached to the anchor GameObject as children. Instead of synchronizing global coordinates, the solution synchronizes the local coordinates relative to the parent,

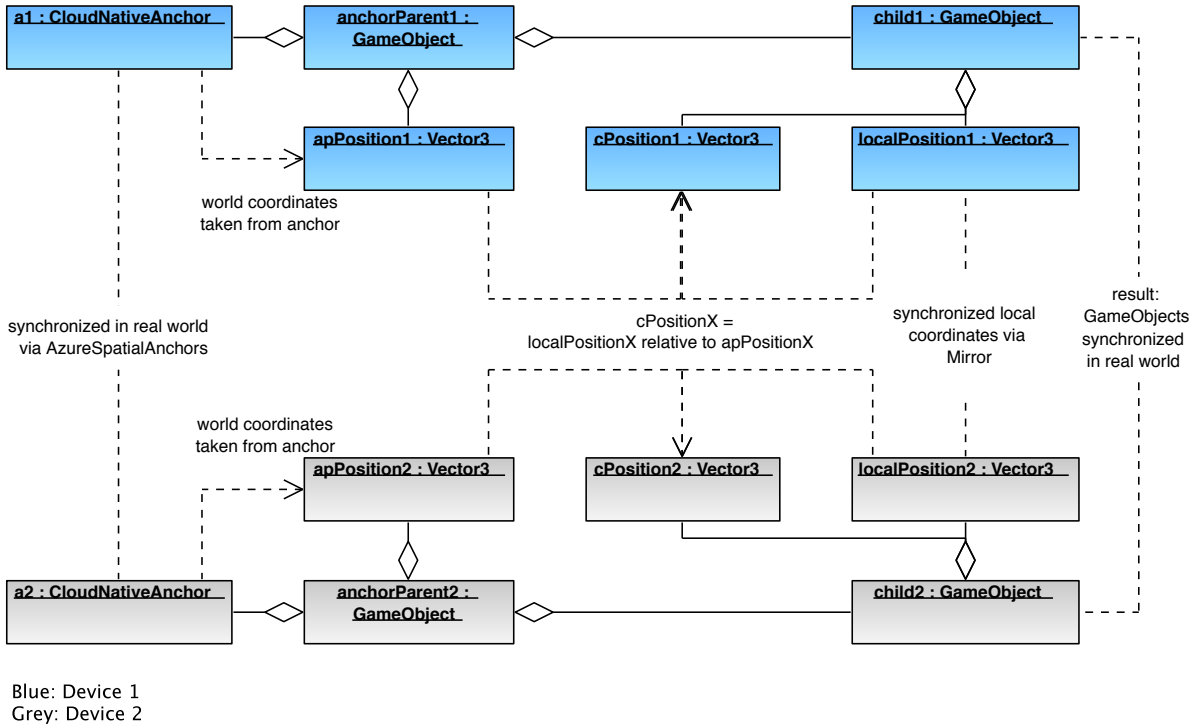

Figure 5.10: Object diagram: implemented concept of pose synchronization for GameObjects across multiple AR sessions.

which is respective to the anchor. This way, GameObject poses can be held consistent in all AR views by having different global coordinates, but synchronized local coordinates and a parent located at a consistent position in the real world. Figure 5.10 depicts the concept exemplary at a scenario with two participants.

5.4 InputControl and InputView: Handling user interaction

The component InputControl handles user input on the AR scene and derives data manipulation instructions for the business logic component ModelLogic from it. The component's main class InputControl executes an update method each frame reading out eventual user input. This concept could be replaced by a more elegant, event-based approach. The continuous query approach, however, was easy to implement and has no measurable impact on application runtime. This thesis' requirements name handhelds and HMDs as supported device types. User interaction with handheld mainly executed via finger touches. The considered HMD is the Microsoft HoloLens. Users can control the HoloLens via gaze tracking for controlling a cursor. The cursor can interact with virtual objects via tracked hand gestures or voice commands. All supported devices and systems are supposed to provide the same collection of data manipulation instructions derived from the product functions stated in Chapter 4:

- Create GameObject
- Select/Deselect GameObject
- Delete selected GameObject
- Move selected GameObject

- Rotate selected GameObject
- Scale selected GameObject

Figure 5.11 shows the component's class structure. Each time the Update method is called on InputControl, it calls the Perform method on each item in a collection of different TouchGesture objects. There is one class implemented for each considered kind of gesture. The Perform method consists of two parts. First, the method GesturePerformed tests, if respective gesture was performed during the frame. If so, the method PerformOperation decides, which data manipulation command with which parameters is called on ModelLogic. Both methods are defined in the abstract TouchGesture class and have to be overridden by each inheriting class. The solution includes seven implemented classes testing on touch gestures for handhelds.

- **TapTG:** A single tap on a virtual object is detected. This input selects or deselects the hit object.
- **DoubleTapTG:** Two taps are detected quickly one after another hitting an GameObject or a detected surface. This input commands the instantiation of a new GameObject at the hit position.
- **PressTG:** A finger is pressed over a longer amount of frames on a single position. This input is not used yet for any model interaction.
- **DragTG:** A single finger touch is wandering from one position to another over time and executes a drag and drop. If the drag begins on the selected GameObject, this input is interpreted to a movement command. In case of the drag beginning next to the selected GameObject, the input is interpreted to a rotation command. The rotation is executed around the y-axis in the direction respective to the finger movement.
- **FlipTG:** A flip is considered to be a quick drag. This gesture was originally intended to perform the deletion of a game object. Tests came to the result, that dragging and flipping are easy to be mistaken. FlipTG is thererfore no longer used. GameObject deletion is executed by pressing a visualized delete button within the scene.
- **PinchSpreadTG:** Two finger touches are detected getting nearer to each other or farer from each other over time. This input commands scaling the selected GameObject.
- **RotateTG:** Two finger touches are detected moving around in a circle over time. Tests came to the result, that this gesture was easily mistaken by the pinch and spread gesture. This is why RotateTG is not used anymore. The rotation is performed via dragging.

Screen touches for each frame can be read out via Unity's Input class. Touch objects have 2D coordinates representing the touchscreen position. These coordinates can be used to calculate a raycast into the 3D scene starting from the camera position and following the touch direction. Raycasts can either be shot on GameObjects or on so-called trackables representing detected surfaces. Raycast execution is handled by the self-developed helper class Raycast.

The component's class structure can easily be extended for HoloLens-specific input detection classes. As the author did not get the opportunity to work with the HoloLens during this work, the implementation of those is missing yet.

The class InputView enhances the AR view with feedback on user input. For each screen touch the class generates a raycast into the AR scene and visualizes the target with a white

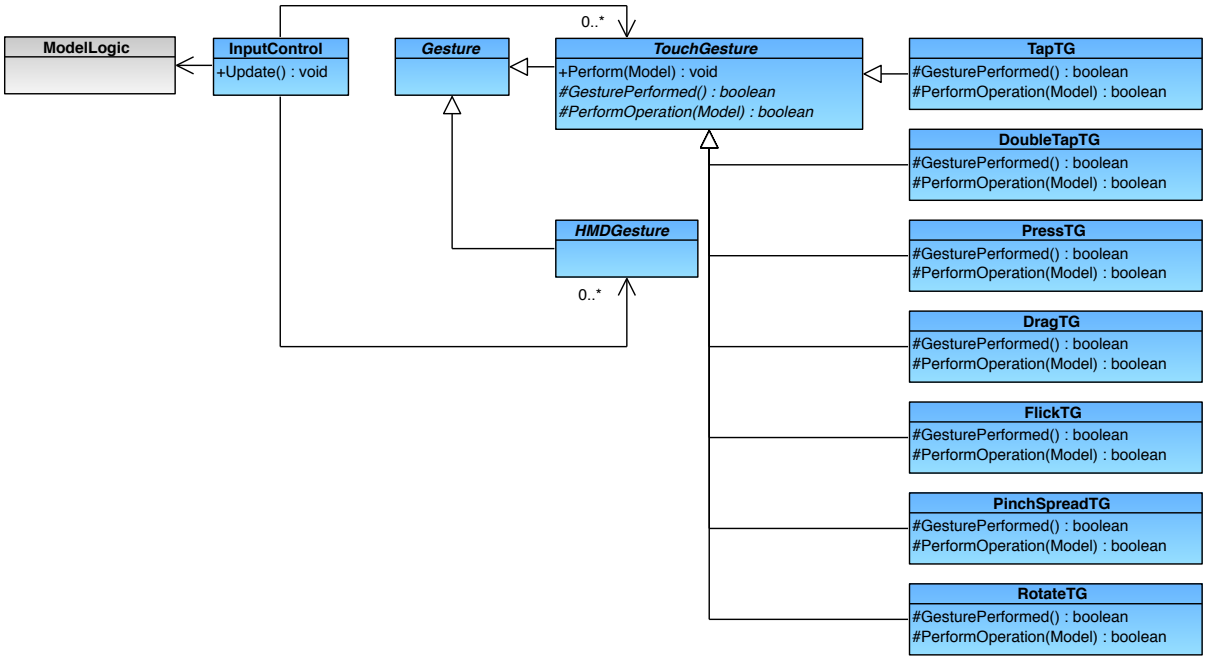

Figure 5.11: The class structure of the InputControl component.

circle. Selection is visualized by a panel floating above the selected GameObject. The floating panel includes a button with a trashcan symbol. In Unity, buttons within 3D scenes can be pressed as well as in two-dimensional ones. The trashcan button triggers the command to delete the selected GameObject.

The interaction of InputControl, InputView and ModelLogic was originally intended to follow the the Model-View-Controller pattern, as depicted in Figure 5.12. Both Controller and View have access to Model data. The Controller accesses the model for data manipulation, while the View accesses it for the purpose of data visualization. In case of updated data, the Model notifies all items in the collection of registered View variants. The notification triggers the update of model data visualization. In a similar way, the View component notifies Controller variants in case of certain events, for example when a cursor position enter a GUI panel. The Controller can then access further information from the view component.

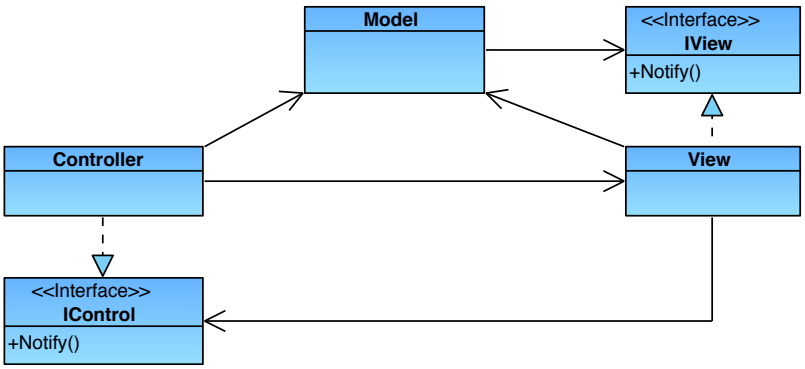

Figure 5.12: The Model-View-Controller pattern.

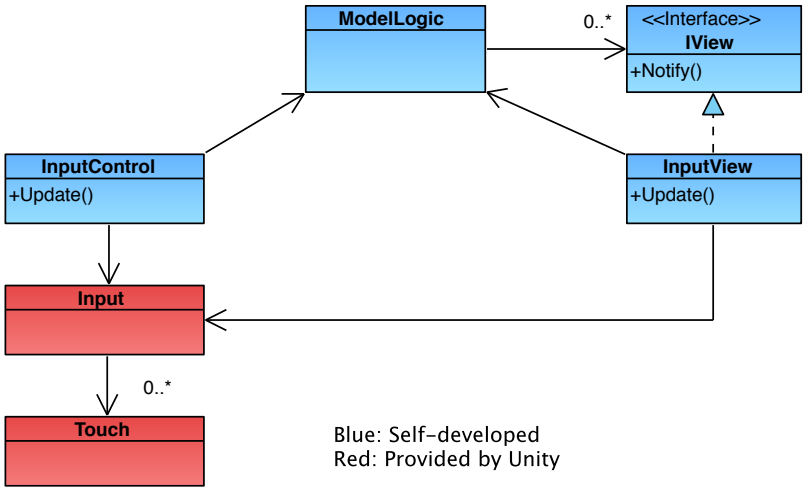

Figure 5.13: The implemented correlation between ModelLogic, InputView and InputControl.

While the solution implements Model interaction according to the MVC pattern. interaction between Control and View are implemented in a different way. this is due to the fact, that the InputControl component is reading out user input continuously instead of an event-based approach. Informations necessary to read by the InputControl are accessed from Unity's Input class solely. Therefore, the InputControl does not require access to the InputView and InputView does not need to notify any Control components. To visualize screen touches, the InputView also requires access to the Input class, too. Figure 5.13 depicts the modified concept.

5.5 ModelLogic: A generic foundation for domain-specific data and business logic

The Class ModelLogic is the main class calculating business logic . The InputControl component calls methods on client-side on it to request data manipulation. After the request is validated and processed within ModelLogic, eventual data manipulation is the requested on server-side via server command. In Unity, the main data of the 3D model is held within GameObjects and their components. The Transform component, for example, holds information concerning position, rotation and scaling of game objects. But how can custom fields be defined for holding further metadata? The MonoBehaviour components are customizable scripts, in which developers can define fields as well as methods. Derived from the MonoBehaviour component is the developed SyncDataContainer class. Each GameObject, that is used to be synchronized in any XshARe application is supposed to have a SyncDataContainer attached. Of course, this can also be a new class inheriting from SyncDataContainer. This new class can be used to define additional metadata. An already existing example for this is the RoomSDC. The class is attached to an empty GameObject to hold a single integer field: the room number, an ID to request the right mapping and anchor data from the Microsoft Azure web server. Figure 5.14 depicts the characterized class structure.

All fields within MonoBehaviours, that are supposed to be synchronized across connected devices, need to be defined with the [SyncVar] header. Mirror provides further components, which need to be attached to GameObjects to synchronize specific GameObject features. For example, the component NetworkTransform synchronizes global coordinates for position, rotation

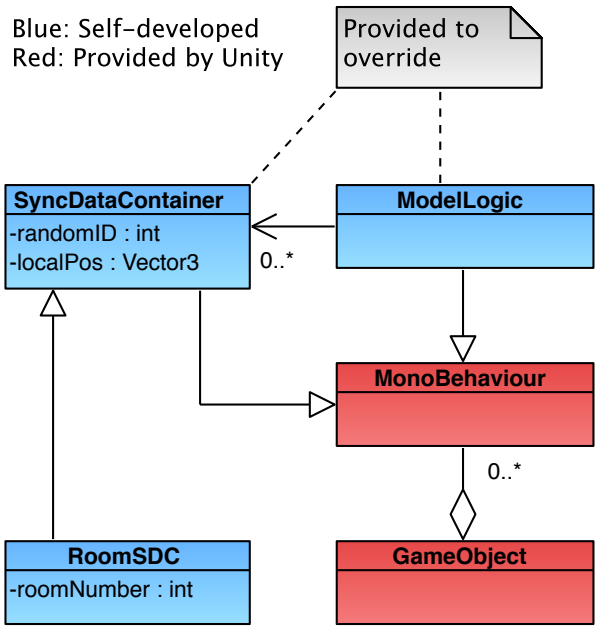

Figure 5.14: The class structure of ModelLogic and SyncDataContainer.

and scaling. As already mentioned, the global coordinates are not the ones to be synchronized, though. What is required, is a way to synchronize local coordinates only. Mirror provides the component NetworkTransformChild for this reason. However, GameObjects with a NetworkTransformChild need a parent with an enabled NetworkTransform to work. UNet, which was working similar to Mirror in that point, provided the ability to disable global synchronization on the parent and still enable local synchronization on the children. Mirror does not provide this feature, however. The solution requires a workaround for this problem. It is overcome by saving the local coordinates within each SyncDataContainer. Changing coordinates leads to manipulating these coordinates. Like every MonoBehaviour, the SyncDataContainer has an overridable Update method, that is called once every frame in Unity. The method is overridden to write the local coordinates from the SyncDataContainer into the GameObject's Transform component. This way, local coordinates can be synchronized with global coordinates kept asynchronous, which is requirement to keep AR views consistent on each device.

6 Evaluation

The presented example applications serve the purpose to evaluate the developed framework in regard of the stated requirements and the initial research question:

How to develop a multi-user AR framework, that can be used for sample applications, which support collaboration and usability.

After Section 6.1 presents two example projects as case studies, Section 6.2 looks back on the project's initial requirements and validates, if and in what extend they are met. Section 6.3 then takes a look especially on the usability aspect of developed applications. This done exemplary with the help of the mill game application, since it provides more complexity in user interaction than the furniture placement example.

6.1 Case Studies

The XshARe framework was used to implement two example applications as case studies: A multi-user AR mill game (nine men's morris) and a furniture placement application. Section 6.1.1 describes implementation and functionality of the mill application, while Section 6.1.2 presents implementation and functionality of the furniture placement application.

6.1.1 The Mill Game Application

The mill application places an interactive 3D board directly on the shared anchor. In contrast to game pieces, the board's pose is not supposed to be manipulated during session. This is why a synchronization of the board GameObject is not required. Each application has its "own" board, which is still viewed consistently in the real world, because the shared anchor is. Placed on the board's points are little invisible spheres with. Each sphere holds a Collider component, which is applicable to detect raycast hits. Attached sphere GameObjects are named with two indices from -3 to 3 to be identified and managed easily by business logic. This way, sphere (x,y) can be associated with the respective point on the board consistently on all devices without dynamic synchronization. Figure 6.1 shows the complex prefab of the game board.

The is game for 2 players. One is playing with white pieces, one with black ones. Each player's possible interactions with the board are dependent on the player's current state, the so-called MillState. At the beginning, when new participants are joining the session, their state

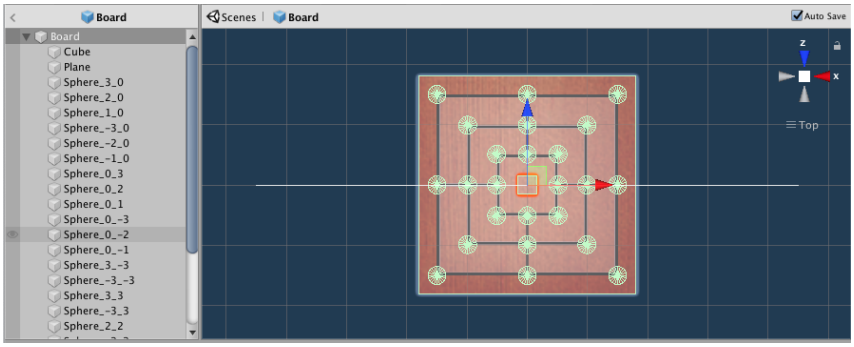

Figure 6.1: The complex prefab of the game board.

is initially set on "Observe". This means they are not allowed to manipulate any global model data. Theoretically, there is no specific upper limit for the amount of participants, who join as observers. When the second participant has joined the session, though, the first one gets assigned the role of the white player, while the second one gets the role of the black player. As the white player is allowed to start, white's state changes from "Observe" to "Place piece", while black's state changes to "Wait", which still means no allowed board manipulation. As long, as a player has placed less than 9 pieces on the board, he can use the interaction to create a new GameObject, which is a cylindrical stone mill stone in this case. The GameObject can only be placed at a position of a colliding sphere and only, if the position is still unoccupied by other stones. In other words, when working with a handheld device, the active player has to perform a double tap the touch screen. This will trigger a raycast shot int the AR scene. If it hits the intended sphere, a cylindrical GameObject will spawn there. Three game pieces of the same color in a row are called a mill. In case of creating a mill, the active player may remove one of the enemy's game pieces. In other words, he is allowed to delete a selected GameObject associated with the enemy player. GameObject selection and deselection is allowed at any time for all participants, even for observers. When player X has finished his turn, his state is set on "Wait", while player Y is the active player now. If he has already set 9 game pieces in the past, his state is set on "Move Piece". Moving own pieces is allowed to adjacent to adjacent empty tiles only. If there are no possible moves left for a player or he has less than three pieces left on the board, his state is set on "Lost" and the enemy's state automatically set on "Won". The game is over. Figure 6.2 depicts the state transitions.

To apply the game rules, the class MillLogic replaces the inherited ModelLogic class within the scene. MillLogic overrides multiple methods to validate data manipulation requests according to the rules. This concerns the following requests:

- Create GameObject
- Move selected GameObject
- Delete selected GameObject
- Select/Deselect GameObject

The methods to scale and rotate GameObjects are not required. disabling unwanted interactions can easily be implemented by overriding the respective method by an empty one. To represent the points on the board, there is the tile class. Objects of this class and their data do not require to be synchronized, as the points on the board remain static throughout the game.

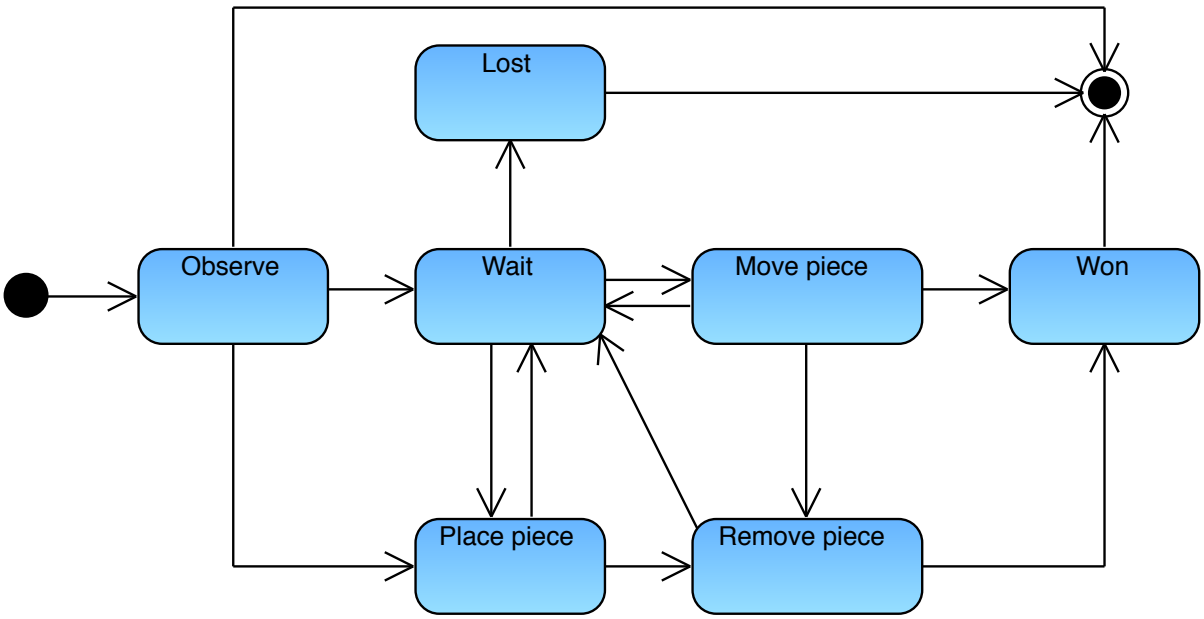

Figure 6.2: State transisitions within the mill game.

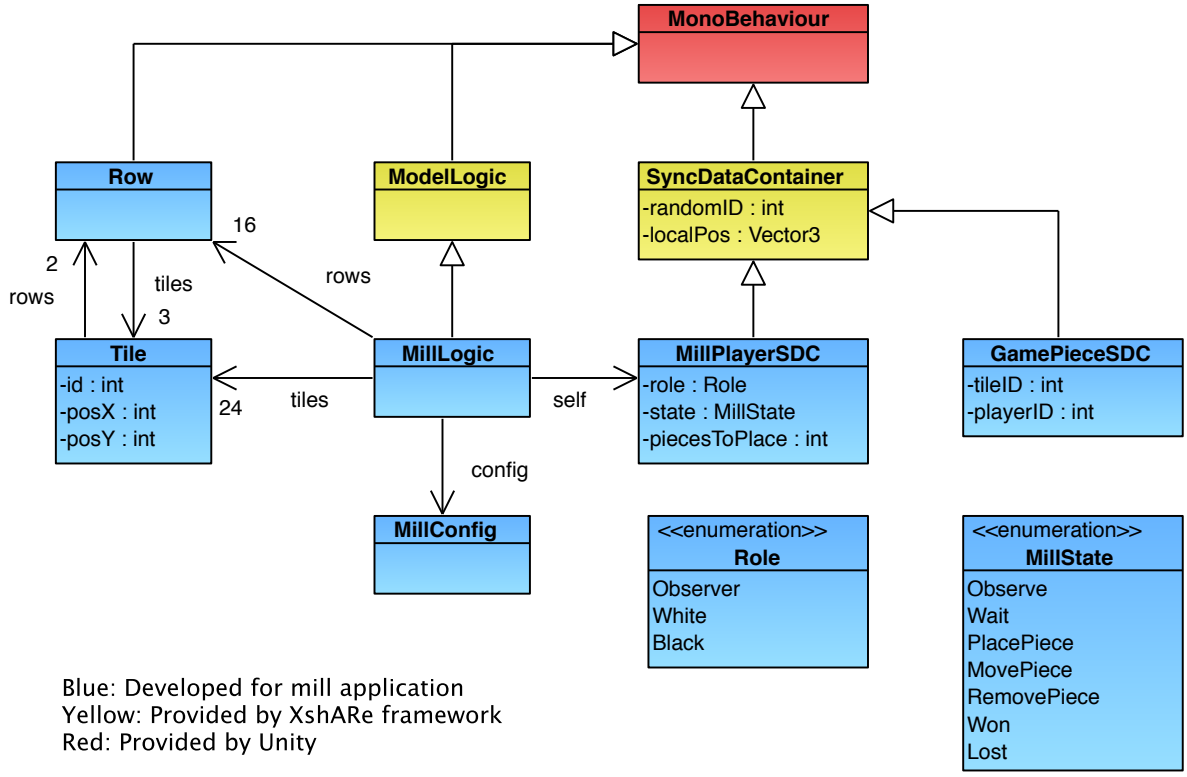

Figure 6.3: The class structure of the mill project.

Therefore, the Tile class does not require to inherit form the SyncDataContainer class. It holds X and Y coordinates to be linkable to the respective sphere on the board GameObject. Furthermore, each Tile object holds unique ID to be identifiable. This is necessary, because pointers on objects are not synchronizable via mirror. A game board holds 24 Tiles. Each tile is part of

two connected Rows, one vertical and one horizontal. A row consists of three Tiles. The class MillPlayerSDC represents a session participant. It holds information about the player's role, state and the number of pieces left to place. The MillLogic references one particular MillPlayerSDC class to represent the own user on each device. Each created GameObject representing game piece holds a GamePieceSDC as component. It contains a tile ID for the tile, it is located on, and a player ID for the owning player. This way, missing object references are replaced. The MollConfig script loads and provides all mill-specific prefabs. Figure 6.3 depicts the class structure of the mill project.

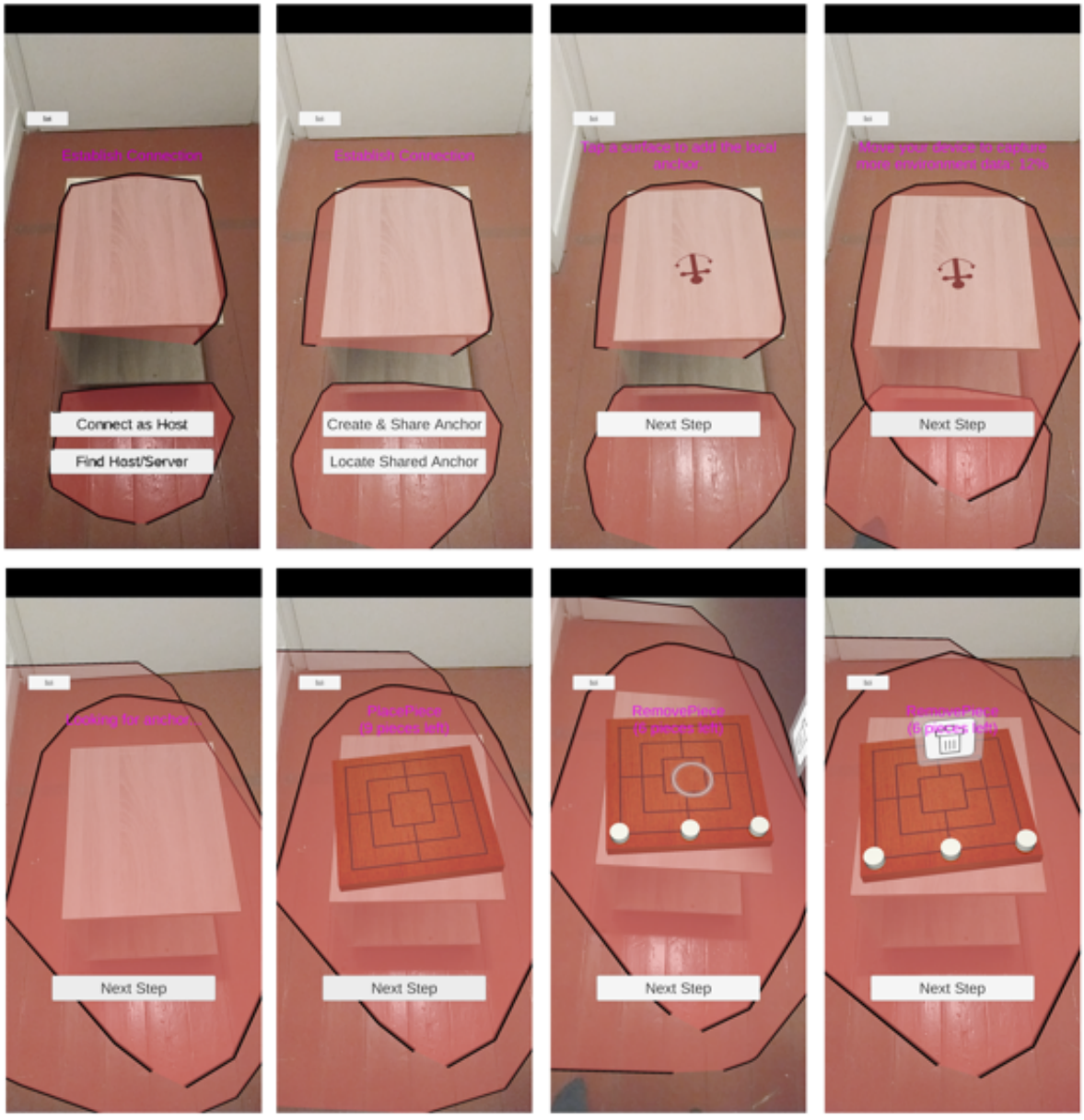

Figure 6.4: Preparation and execution of the AR mill game.

Figure 6.4 shows the process of preparation and user user interaction with the virtual mill board in eight screenshots beginning with the top left and ending with the bottom right. Note, that the preparation phase is taken from the Azure Spatial Anchors Examples, except the

network connection menu:

1. To establish a network connection, choose between host and client role.
2. Create a new anchor for the session, if not already done.
3. Tap on a surface to set anchor pose. The transparent virtual surface with black border represents a detected real world surface as feedback. In this case, the tabletop and parts of the floor were detected as horizontal surfaces. The anchor symbol is a GameObject representing the anchor's pose.
4. To upload the anchor, the room has to be scanned for mapping data. When the visualized percentage number is at 100 percent, data is uploaded automatically to the Microsoft Azure servers.
5. After the anchor is created and uploaded, it can be located by all participating devices. This requires a phase *Looking for anchor...* .
6. After the anchor is located, the multi-user AR session can start. In this case, this is the mill game. The virtual mill board spawns exactly at the position of the anchor.
7. Players can place game pieces on the board with double tap. When only one user is taking part, he can place as many stones as he wants. A white circle gives feedback on finger touches.
8. Stones can be selected vie single tap. The selection is visualized via a selection panel including a deletion button. The button can be used for deleting the GameObject. Selected GameObjects can also be moved via drag. Scaling and rotating is disabled for game pieces in the mill game

6.1.2 The Furniture Placement Application

A second developed example is an application for collaborative furniture placement. Furniture placement apps on handhelds have become popular provisions by furnishing groups. Customers can place virtual equivalents of buyable furniture inside their home. This way, they can try out different furnitures in regard of suitability. An example for furniture placement apps is the IKEA Place app[1]. The application lacks multi-user support and therefore makes collaboration difficult.

Compared to the mill example, the developed furniture placement application is a much less complex. The application uses almost all standard types of provided data modification. The only prohibited one is the scaling, because it is necessary for the virtual furnitures to keep their realistic size. The request to create a new GameObject lets a panel appear in the AR scene, which includes a button for every furniture type to choose. This panel is not synchronized and therefore only viewed on the creating device. At the moment, there are three types of furniture choosable: wooden chairs, tables and shelfs. Only very few additional program logic was necessary to be developed for the furniture placement application, as most of it is already implemented standard within the developed and used XshARe framework. GameObjects representing furniture only require to have a simple SyncDataContainer attached. Figure 6.5 shows all included virtual furnitures and the selection panel in a single 3D scene.

[1]IKEA Place App - https://www.ikea.com/ch/en/customer-service/mobile-apps/ikea-place-app-pub0bab12b1

Figure 6.5: Selection panel and furniture of the furniture placement application.

Figure 6.6 shows all interactions possible interactions within the furniture placement example application from the top left to the bottom right:

1. The session is starts after the anchor is located. In this case, no additional GameObjects are spawning initially, except the standard anchor symbol.
2. A double tap creates a selection panel. This panel is not synchronized via network and therefore visible only on the creating device. This GameObject cannot be selected or manipulated either.
3. Tapping on the table button lets the selection panel disappear in exchange for a virtual table with the exact same pose.
4. Tapping on the furniture GameObject selects it. This is visualized by the selection panel above it including the deletion button.
5. Dragging with the finger across the screenshot rotates the selected GameObject, if the dragging did not start with the finger on the selected GameObject.
6. If the dragging does start on the selected GameObject, it is moved to the dragging's end position.
7. Users can create and manipulate various virtual furnitures within the AR scene.

6.2 Evaluation of Requirements

The two developed example applications were used as case studies to evaluate the stated requirements.

- **R1: Supporting multi-device sessions**
  Both developed example applications connect multiple devices with each other. The mill variant applies user-specific rules. The is applied by a network connection between the devices and sharing information between them via the third-party network component Mirror.
- **R2: Supported cross-platform range**
  The applications are tested successfully on Android and iOS devices. Support for HoloLens

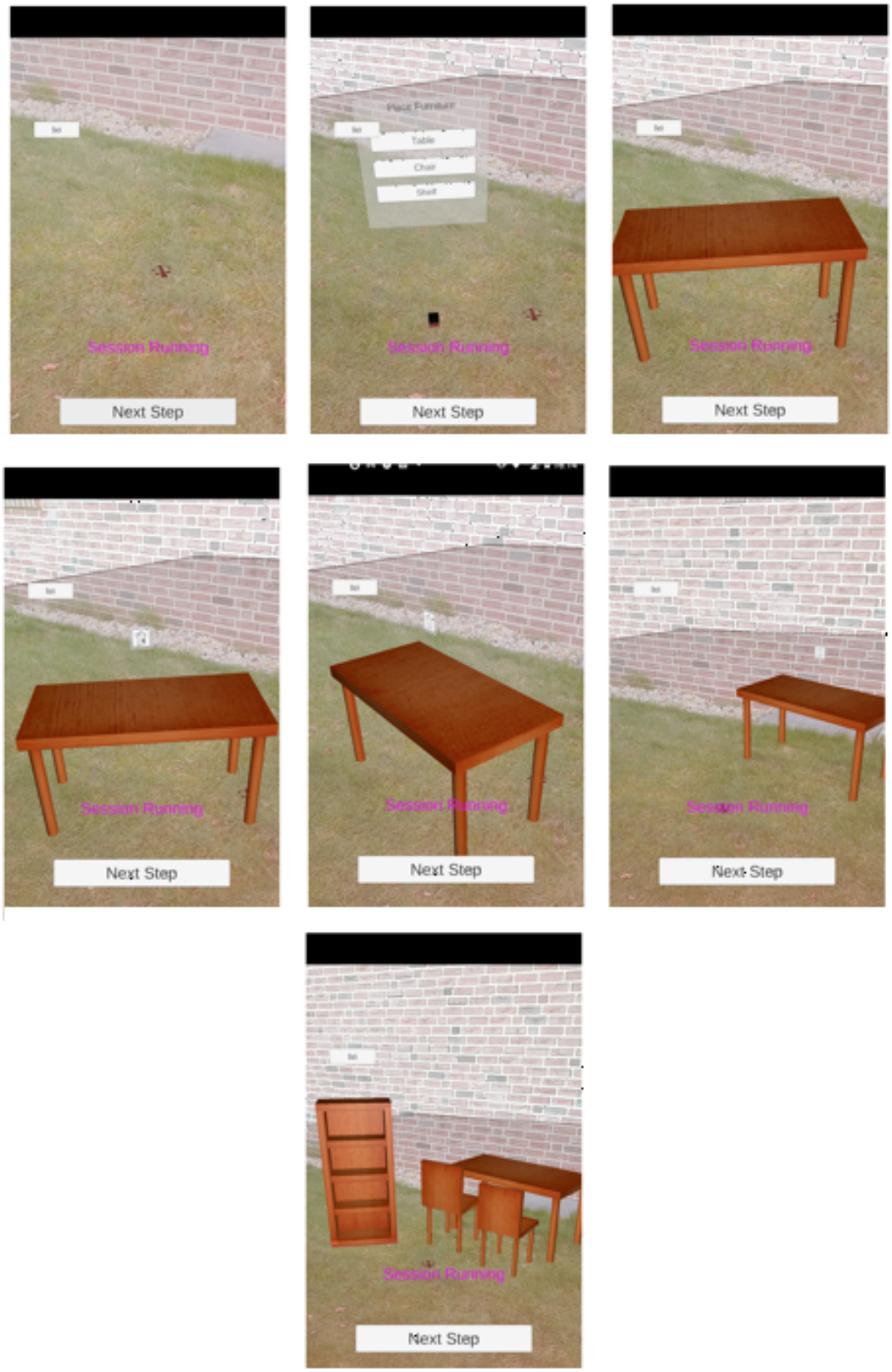

Figure 6.6: Interactions within the furniture placement application.

can prospectively be achieved by extending the framework. The biggest challenge to overcome, when the HoloLens is supposed to be integrated, is expected to be the integration of HoloLens' interaction modalities. Unity's huge range of supported platforms is the guarantor for cross-platform support with a single software project in use.

- **R3: Supported device types**
  The framework's considered interaction modalities only include screen touches at the moment. Therefore, handheld devices are the only devices supported at the moment. Other challenges than handling user input are not expected for integration of HMDs. The integrated Azure Spatial Anchors Examples, however, do provide HMD support.

- **R4: Consistent AR view**
  The AR view is consistently across all connected devices. This is achieved by using Azure Spatial Anchors as shared reference points in the real world. All virtual objects' poses are considered relatively to the reference point. Their local coordinates relative to the anchor are synchronized via network connection, which is sufficient to keep the AR scenes consistent, while the objects' global coordinates remain asynchronous to each other.

- **R5: Co-located AR**
  Developed applications work in co-located AR only, as the anchor sharing requires comparable mapping data from each participating device.

- **R6: Markerless AR**
  The framework's used concept is to be classified as co-located, markerless AR. Using physical markers instead of the described approach to locate reference points is not supported.

- **R7: Interactivity for multiple users**
  All participating users are able to interact with the AR scene through their own device. Individual interaction abilities for single users or specific user groups can be modified in developed applications. The mill game is a good example for that. While observers have no option to manipulate the AR scene, the two players can change the state of the board according to the applied rules.

- **R8: Framework character**
  As the two example application have shown, the developed main project can be used as a reusable foundation for multi-user AR applications. It does not only have a library character providing a couple of services on demand. Instead it is as full Unity project already ready to be compiled and run on devices. The extensions necessary for individual domain-specific applications can be seen as a component integrated in an existing framework.

6.3 Evaluation of Application Usability

An extra evaluation study was conducted to examine usability of applications made with the developed XshARe framework. The evaluation was based on the mill game application. Section 6.3.1 describes the approach in detail, while Section 6.3.2 takes a look on the evaluation results. Those results are then interpreted in Section 6.3.3. Section 6.3.4 takes a look on possible threats to research validity. A conclusion in Section 6.3.5 finishes the evaluation.

6.3.1 Evaluation Approach

A main aspect of the work is collaboration. This is why the evaluation approach is focussed on the question of usability and practicability of multi-user AR applications in a collaborative scenario. The hypothesis of the study is:

Using multi-user AR applications is practicable to enable collaboration.

To play a session of the AR mill game, at least two participants are required. This is why multiple teams of two were invited to prepare and play together. Each team played the AR variant once and an analogue variant once. Figure 6.7 shows both the analogue and the virtual mill board. The setup can be summarized as followed:

- Eight teams of each two participants.
- Six female, ten male participants.
- Six teams tested in university, two in living rooms.

Figure 6.7: The virtual and the analogue mill board to play with during the evaluation study.

To be enable a session, there are specific requirements to the surrounding. Room requirements also have to ensure comparable sessions.

- The session needs to take place in a bright room.
- The room needs access to a WIFI network with permission to connect devices.
- The room needs a rectangular table with sharp edges to ensure a robust surface detection.

The following tools and devices were used during the sessions:

- User 1 used a Motorola moto g8 power as handheld device.
- User 2 used a Samsung Galaxy A7 as handheld device.
- Both users used an analogue mill game.

Each team executed two variants in varying order. Each variant consists of several tasks, that the participants have to complete.

- **Variant 1: Analogue mill game**
  - Task 1: Build up the game board.
  - Task 2: Play the game for five minutes.
- **Variant 2: Multi-user AR mill game**
  - Task 1: Connect the devices.
  - Task 2: Set and upload a Spatial Anchor.
  - Task 3: Locate the uploaded Spatial Anchor.
  - Task 4: Play the AR game for five minutes.

The sessions were monitored and measured by three metrics.

- Effectiveness
- Efficiency
- User satisfaction

The effectiveness relates to completion. This includes whether all users could execute their sessions to the end, whether they could complete all tasks and which technical issues appeared during the sessions. Efficiency relates to measured times. For both variants, times were measured for game interactions: Set stone, Move Stone and Remove stone. Furthermore, preparation time was measured for each variant. The analogue variant requires building up, while the AR variant requires network connection, anchor creation and anchor location. User satisfaction was measured by a survey, which each participant had to fill out after his session each for both variants. The questionnaire includes 17 items, of which the first ten are derived from the System Usability Scale (SUS), a simple scale to measure usability of information systems [Bro96].

- **Q1:** I think that I would like to use this game frequently.
- **Q2:** I found the game unnecessarily complex.
- **Q3:** I thought the game was easy to use.
- **Q4:** I think that I would need the support of a technical person to be able to use this game.
- **Q5:** I found the various functions in this game were well integrated.
- **Q6:** I thought there was too much inconsistency in this game.
- **Q7:** I would imagine that most people would learn to use this game very quickly.
- **Q8:** I found the game very cumbersome to use.
- **Q9:** I felt very confident using the game.
- **Q10:** I needed to learn a lot of things before I could get going with this game.

Questions **Q11 - Q17** were derived from the stated requirements:

- **Q11:** I found working with my device more or less attractive than with my partner's device **(R1- R3)**.
- **Q12:** I had a common feeling of virtual objects' positioning with my partner **(R4, R5)**.
- **Q13:** I found interaction with game elements comfortable and intuitive **(R7)**.
- **Q14:** I had the feeling to see exactly the same as my partner **(R4, R5)**.
- **Q15:** I found preparing and set-up for the game quick, comfortable and easy to understand **(R6)**.
- **Q16:** I had an overview about my partner's assets and game situation **(R1, R4)**.
- **Q17:** I found it easy to follow the rules and play the game as intended **(R1, R7)**.

6.3.2 Evaluation Results

Now that the evaluation approach is presented, their results are discussed in terms of the three considered metrics: effectiveness, efficiency and user satisfaction.

Effectiveness

In regard of effectiveness, the evaluation sessions were predominantly successful. All preparation phases were completed successfully during all AR sessions, so that all participants were able to take part in the AR mill game. During the game phase, all three types of game interaction, setting, moving and removing were executed successfully by all participants. Several technical problems occurred during some of the sessions though, that complicated one or the other session:

- The surface detection failed during one session on one device. This lead to the disappearance of the virtual game board, since the shared anchor working as reference point could not be found in the room any more. This accident occurred after more than five minutes of playing, though.
- Only rectangular tables were found reliably as surfaces. An experiment with a round table ended without a detected table top.
- Light was also complicating factor. Dimmed light in a room made scanning harder. The recommended lighting intensity in the room is daylight-like or comparable.
- Several teams did not get to moving a stone within 5 minutes of playing. This is because both players first have to set nine game pieces each, before they can move stones. In several cases, the measured five minutes were over before that. The participants were allowed to keep on playing anyway, even if the times for interactions were no longer noted.

Efficiency

Looking on the numbers of overal game interactions within five minutes shows a huge difference between the AR variant and the analogue real world variant. With 323 measured interactions the analogue variant included twice as many interactions than the AR variant with only 152 measured interactions. This implies a slower interaction with the handled devices than with the real world. Predominant interaction type during the AR session was stone setting, since moving and removing are actions, that take place later in the game. This is why the proportion of moving and removing is much higher during the real world variant. Figure 6.8 shows the results

of measured game interactions in total.

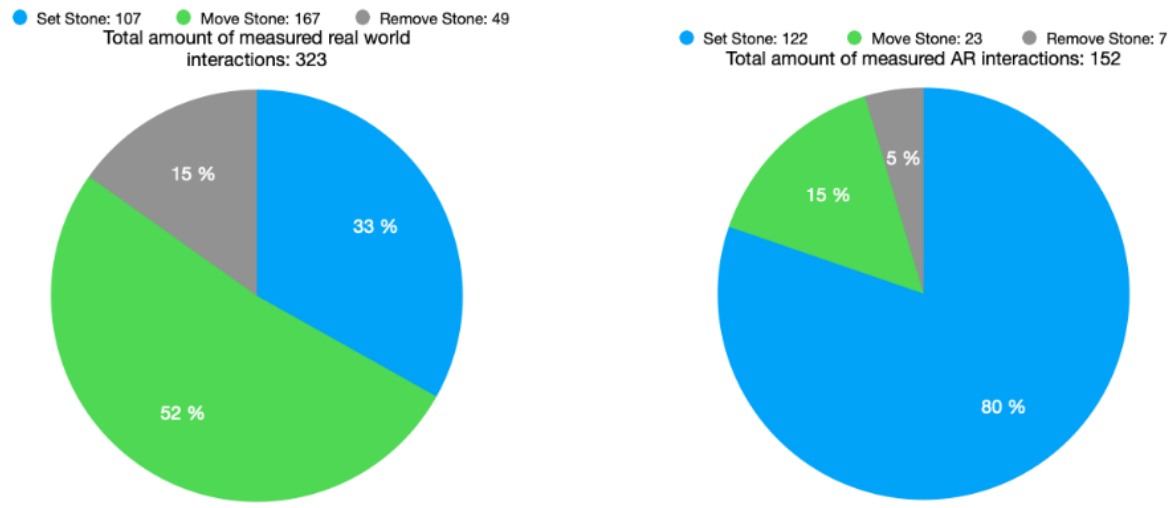

Figure 6.8: The results of measured game interactions in total.

During the preparation phase of AR session, anchor setting took the longest average time with 75.2 seconds. In contrast, network connection was established after only 10.1 seconds and the anchor was located after 36 seconds on average. This results in an average total preparation time of 121 seconds for the AR application in contrast to only 27 seconds average time for building up the real world mill game. It is important to say though, that the AR preparation phase is generic and independent from the actual AR use case. That means, in other scenarios with a longer preparation time for the real world scenario the two minutes of preparation for an AR alternative could possibly compete better in comparison. Figure 6.9 shows the results for average preparation times.

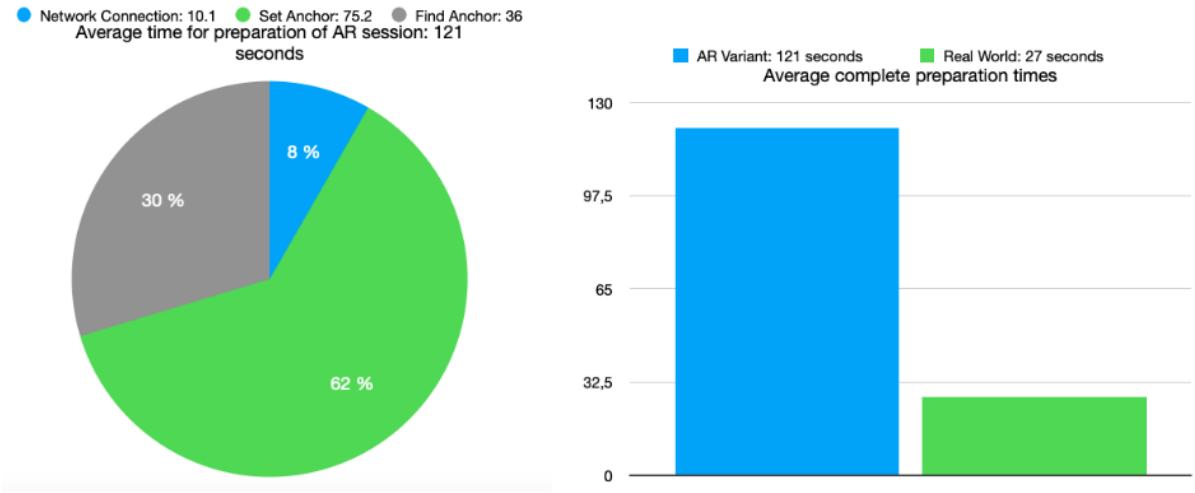

Figure 6.9: The results for average preparation times.

As already indicated, the average time for a single game interaction takes longer in the AR

variant than in the analogue variant for all interactions. On average an AR interaction took 16.4 seconds, while the average real world interaction took 5.4 seconds. These times include time for overthinking the next step. With 10.1 seconds to 4.4 seconds, setting stones in the AR mill game only took round about twice as long as in the real world variant. Moving and removing stones, however, took much longer. This is because multiple cases of these interactions took up to two minutes for participants having trouble to select the intended game piece, hit the deletion button or hit the selected stone for moving. Figure 6.10 shows the comparison of average game interaction times.

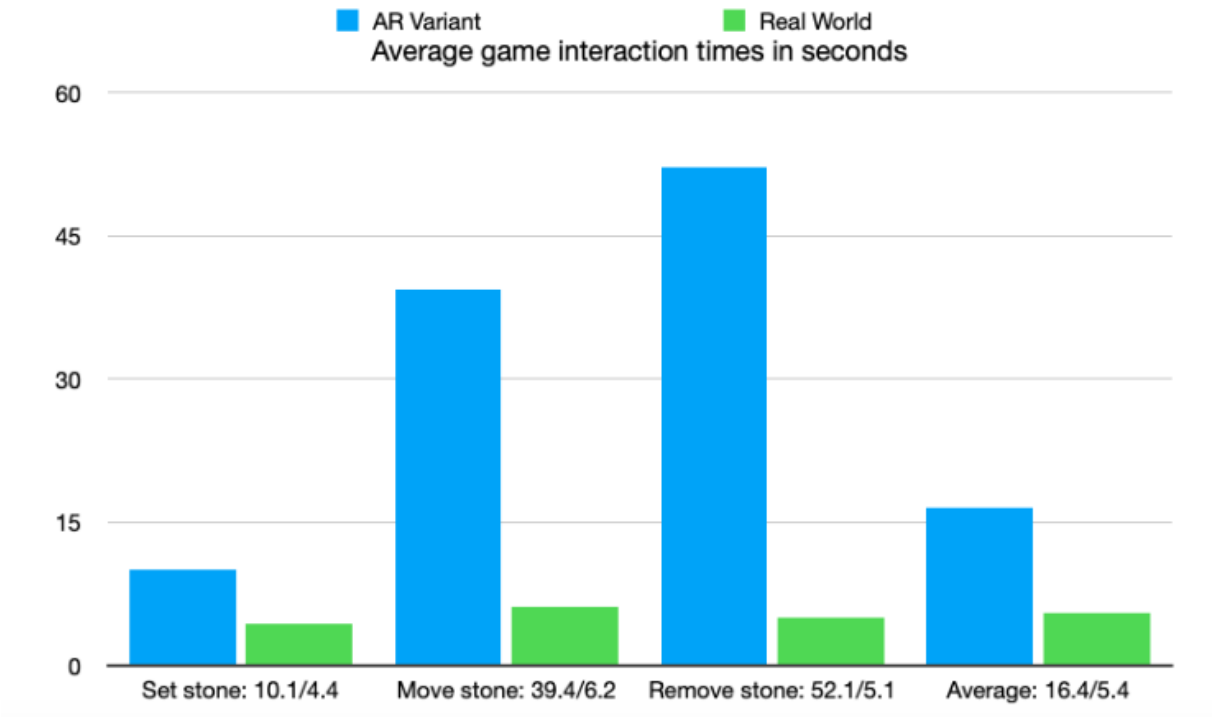

Figure 6.10: The comparison of average game interaction times.

User Satisfaction

Each item of the survey was answered with a value from 1 (Strongly disagree) to 5 (strongly agree). Depending on whether the items are formulated positively or negatively, their result values can be converted to a value between 0 and 4. Adding the average values multiplied with 2.5 returns a SUS-score between 0 and 100. The calculated SUS-score for the analogue variant is expectably high. For the first ten items directly derived from the SUS, the score is 88.3, while it is 85.1 for the last seven items. The calculated score for the AR variant is notably lower. While the first ten items reach a SUS-score of only 59.7, the last seven reach a score of 69.1. Figure 6.11 gives an overview of all determined SUS-values for the items **A1** to **A17**.

6.3.3 Interpretation

A closer look on the single items shows that there are items with large differences in rating as well as items with only small differences.

- **Low difference ($<$ 0.5):**

Real World Variant	SUS-Value	AR Variant	SUS-Value
A1	2,6875	A1	2
A2	3,875	A2	2,6875
A3	3,8125	A3	2,375
A4	3,8125	A4	1,75
A5	3,4375	A5	2,625
A6	3,5625	A6	3
A7	3,625	A7	2,625
A8	3,4375	A8	2
A9	3,5625	A9	2,875
A10	3,5	A10	1,9375
A11	2,625	A11	2,25
A12	2,9375	A12	3,3125
A13	3,8125	A13	2,4375
A14	3,6875	A14	3,4375
A15	3,6875	A15	1,75
A16	3,875	A16	3,3125
A17	3,6875	A17	3,25
Real World Variant	**SUS-Score**	**AR Variant**	**SUS-Score**
A1-10	88,28125	A1-10	59,6875
A11-A17	85,09375	A11-A17	69,125

Figure 6.11: All determined SUS-values for the items **A1** to **A17** and calculated scores.

Q11: I found working with my device more or less attractive than with my partner's device

Q12: I had a common feeling of virtual objects' positioning with my partner

Q14: I had the feeling to see exactly the same as my partner

Q17: I had an overview about my partner's assets and game situation

- **High difference (>1.5):**

  Q2: I found the game unnecessarily complex.

  Q4: I think that I would need the support of a technical person to be able to use this game.

  Q10: I needed to learn a lot of things before I could get going with this gam

  Q15: I found preparing and set-up for the game quick, comfortable and easy to understand.

While the items with low difference in the rating seem to refer to the mill game itself, while the items with high difference in the rating seem to refer to the preparation phase.

6.3.4 Threats to Validity

The validity of the measured and interpreted results is threatened by several aspects.

- **Statistic validity:** Statistic validity can be threatened especially by the low amount of measurements for the interactions to move and remove stones in combination with the variance of measured values. Having a bigger data base on measurements would certainly ensure the calculated average results.
- **Internal validity:** Internal validity is given, when it is ensured that the changes of a dependent variable within the study can be traced back exclusively to the assumed independent variable. Dependent variables are are all measurements in this case, while the dependent variable is playing either the real world variant or the AR variant of the game. To exclude historical threats, half of all teams started playing one variant first, while the other half started playing the other one first. To ensure comparable prerequisites for each session, each team was told the same introduction. A bigger threat to internal validity is the used measurement technique, though. Interaction had to be reported by the test participants to the investigator. The investigator then had to press a button on a developed helping application to stop times. During the sessions, several values were missed to be recorded due to misunderstandings between test participants and investigator. Furthermore, the process from executing an interaction, reporting and recording includes a delay. This delay can make a bigger relative difference on values for game interactions, which only take several seconds time on average. The delay is expected to have the same impact on values in both variants, though, so that validity should not be seriously threatened by this aspect. It was not recorded, what kind of prior knowledge about AR the participants had. This aspect should be weakened heavily, though, by the extensive and detailed insrutction.
- **External validity:** External validity concerns the question if the results can be transferred universally applicable to all people. The majority of test participants was was observed in a room in university. They all were either students or scientific staff working in own AR-related projects. After the introduction, though, they did not seem to have less problems with usability than the subject-unrelated minority. Therefore it can be assumed, that results are transferrable to the population's average.

6.3.5 Conclusion

In conclusion, the solution can be evaluated very usable in regard of effectiveness. All participants could execute all actions and use the application as intended. In regard of time efficiency, the generic interaction modalities provided by the developed framework fall off compared to real world interaction. The standard preparation phase takes about two minutes on average. User satisfaction of the participants was relatively high during the mill game and relatively low during the preparation phase.

In regard of the stated hypothesis it can be said, that collaboration in multi-user AR applications works, but with slower interaction compared to equivalent simple actions in the real world. Usability highly depends on comfortable interaction modalities. The framework's design decisions in regard of interaction, however were not mainly motivated by reusability for the developer. The provided set of interaction types may not be perfect for each case of AR application. In a mill game, for example, setting and removing a stone could also be executed by only a single tap.

7 Summary and Future Work

Now that the evaluation is discussed this chapter can conclude and recap this work. Section 7.1 summarizes the achieved and missed goals of this thesis. An outlook on possible improvements of the developed framework is presented in Section 7.2. The section also suggests ideas for further research and development.

7.1 Summary

This thesis was motivated by the lack of an available cross-platform multi-user AR framework. While there are several applications already available, which provide interactive, markerless, co-located multi-user AR experiences with consistent AR scenes across all participating devices, goal of this thesis was to develop a framework providing necessary and reusable features for multi-user AR as foundation to build domain-specific AR applications on top of it. To do so, the work had to face several challenges:

- Find an AR engine with a wide range of cross-platform support.
- Implement a way to synchronize virtual objects across devices.
- Find a way to integrate these virtual objects into AR scenes with consistent views
- Extend the approach by managing metadata
- Enable business logic computing in this multi-computer scenario.
- Implement ways to interact with the AR scene on different devices.
- Offer an interface to make the framework usable for other developers.

During this thesis the framework XshARe was developed. It provides a foundation for cross-platform multi-user AR applications. The framework includes generic components reusable by a diverse range of domain-specific instances. To enable a consistent AR view across all participating co-located devices, the solution shares anchors as reference points to orientate virtual objects at. The framework is basically a Unity project integrating the Microsoft Azure Spatial Anchors cloud service for anchor sharing and the third-party component Mirror for network connection. Unity provides a huge range of supported systems to run the project on. While the framework is tested successfully for Android and iOS handhelds, the originally intended HoloLens integration

was not implemented due to missing access to a HoloLens device. Therefore, HMD support is still lacking. The framework is designed to be easily extendable, though.

To develop the solution, the work began with research on several aspects. First, foundations were research on Augmented Reality. An overview over supporting device types, IDEs and AR SDKs was researched, too. Further foundation research was done on current multi-user AR cloud services and available network components for Unity projects. Subsequently, other scientific works and software related to multi-user AR were researched.

To evaluate the XshARe framework, two applications were developed with it as case studies: a mill (nine men's morris) game and a furniture placement application. The mill game was used to execute an evaluation study on usability with multiple invited test subjects. Compared was the AR mill game with an analogue real world variant. The results showed that the design of comfortable user interaction modalities are crucial for multi-user AR scenarios to compete with simple real world interactions in terms of usability.

7.2 Future Work

There are several options to improve the developed solution further.

- The missing HoloLens integration could be caught up.
- To improve usability, the existing button menus for session preparation could be reworked.
- Usability for developers using the framework could be improved by implementing framework specific menu items within Unity. For example, menu items for creating new XshARe scenes, synchronisable GameObjects, ModelLogic scripts or SyncDataContainer scripts could be provided.
- More dynamic case studies could be developed with the framework. Unity and Mirror also provide physics simulation. Use of physics is not necessarily trivial in multi-user AR scenes though, since participating devices have inconsistent coordinate systems within their AR scenes.
- If case studies were more dynamic, the continuous dynamic change of data would have to be executed on server-side only. Therefore, it could be worth considering to rework the whole framework architecture and execute all business logic on server-side only.
- With the event system, Unity provides an own event-based concept for interaction with GameObjects. This also includes selection and deselection. Integrating different device specific interaction modalities could be tricky, though. This is why the event system was ignored during this work.
- Moving objects could be implemented with continuous updates on position. This would require a concept for locking and unlocking moved objects for other users over time.

Further research in upcoming works could include the use of data persistency. This work's approach requires scanning and mapping the room and setting a shared anchor each time a new session is started. Mapping data could also be stored for a longer time than just a couple of hours. This way new ideas for AR applications would be possible, for example, virtual notes pinned to a real world refrigerator over multiple days. In fact, persistency is the second wide area of use for shared anchors. The preparation phase could also be shortened massively in time

by scanning the room and setting an anchor only once and then reuse the data over and over for multiple days. This could also be combined with a concept for location-based AR. User positions could be searched via GPS or comparable ways to locate. Whenever the user starts the persistent AR application in a specific room, the GPS location determination could be set in relation to a certain room ID. This ID could then be used to find the right mapping and anchor data within a database. This way, huge building complexes like universities or museums could be captured for instant markerless AR with persistent content for multiple users. Theoretically, Microsoft Azure and the Azure Spatial Anchors Examples should provide all technical features required to implement such a project. Using multiple anchors within a single room are another thinkable option for further research. This work has encountered no problems with orientating all virtual objects to a single anchor, though. Even in cases of longer distances of several meters, like in the furniture placement case, there were no problems with visual inconsistencies encountered. A multi-user AR framework could also use generic and reusable features to establish user roles and rules like privacy rules. Furthermore, case studies could be invented with special features like the use of the user's device position, which is relatable to the active camera's coordinates within the Unity scene. For example, a multi-user AR application is thinkable providing a virtual chatroom with text messages floating over each user's head. Finally, another interesting aspect of multi-user AR to research on could be the use of sound. As handhelds provide speakers, they can not only be used as AR devices, but also as source of sound. A concept could be developed to find out which sound to play in which volume on which device to make a multi-user AR scene more immersive.

Since this thesis has fulfilled its main goal to provide an interactive and consistent AR scene for multiple devices, the options for possible further work are many.

Bibliography

[ATKS] Ms Kamble AL, Ms Shinde TK, Ms Kothiwale, and Khot SS. Real time and distributed computing systems.

[Avr16] Abel Avram. Faas, paas, and the benefits of the serverless architecture. https://www.infoq.com/news/2016/06/faas-serverless-architecture/, 2016.

[Azu97] Ronald T. Azuma. A survey of augmented reality. *Presence: Teleoperators and Virtual Environments*, 6(4):355–385, 1997.

[BCL15] Mark Billinghurst, Adrian Clark, and Gun Lee. A survey of augmented reality. 2015.

[Bri00] Neil Briscoe. Understanding the osi 7-layer model. *PC Network Advisor*, 120(2):13–15, 2000.

[Bro96] John Brooke. Sus: a "quick and dirty'usability. *Usability evaluation in industry*, page 189, 1996.

[Cow17] Ric Cowley. Directive games showcases ar moba the machines at apple keynote. https://www.pocketgamer.biz/news/66592/directive-games-showcases-the-machines/, 2017.

[Dic18] Megan Rose Dickey. Spatial raises $8 million to bring augmented reality to your office life. https://techcrunch.com/2018/10/24/spatial-raises-8-million-to-bring-augmented-reality-to-your-office-life/?guccounter=1, 2018.

[dlP17] Salva de la Puente. Multi-user experiences with a-frame. https://hacks.mozilla.org/2017/10/multi-user-experiences-with-a-frame/, 2017.

[ET17] Ruwan Egodagamage and Mihran Tuceryan. A collaborative augmented reality framework based on distributed visual slam. In *2017 International Conference on Cyberworlds (CW)*, pages 25–32. IEEE, 2017.

[Gaj20] Dejan Gajsek. Unity vs unreal engine for xr development: Which one is better? https://circuitstream.com/blog/unity-vs-unreal/, 2020.

[Glo18] Don Glover. Unet deprecation faq. https://support.unity.com/hc/en-us/articles/360001252086-UNet-Deprecation-FAQ, 2018.

[Gos19] Phil Gosch. Übersicht ar frameworks - 2019 update. https://codefluegel.com/blog/uebersicht-ar-frameworks-2019-update/, 2019.

[HBO05] Anders Henrysson, Mark Billinghurst, and Mark Ollila. Face to face collaborative ar on mobile phones. In *Fourth ieee and acm international symposium on mixed and augmented reality (ismar'05)*, pages 80–89. IEEE, 2005.

[Hou20] Brandi House. Choosing the right netcode for your game. https://blogs.unity3d.com/2020/09/08/choosing-the-right-netcode-for-your-game/, 2020.

[KHLI12] Shunichi Kasahara, Valentin Heun, Austin S Lee, and Hiroshi Ishii. Second surface: multi-user spatial collaboration system based on augmented reality. In *SIGGRAPH Asia 2012 Emerging Technologies*, pages 1–4. 2012.

[Kle07] Sebastian Klepper. Augmented reality-display systems. *Technische Universitaet Muenchen, Munich, Germany, Jul*, 4, 2007.

[Lee18] Nicole Lee. Spatial's collaborative ar platform is basically facetime in 3d. https://www.engadget.com/2018-10-24-spatial-augmented-reality-3d.html?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAH7lVXPb_3O_E9_QDifJoH0KTCehYHLpGCBfpK1SVa_2eTS-AFrBHGGIQtzl4XfkAunuHPo2jKHDz_PjLJNS2Nw-HO44FYEMiqBO325NRBiR5eyxykRz2_6fDEMDtxvf44r9UQNaKlEW_oYj_VCUGGqS3KeN2rFf-0PG1AK5aIc, 2018.

[LNZ+19] Tengpeng Li, Nam Son Nguyen, Xiaoqian Zhang, Teng Wang, and Bo Sheng. Promar: Practical reference object-based multi-user augmented reality (poster). In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pages 531–532, 2019.

[Mat18] Neil Mathew. Building a multiplayer arkit game with placenote sdk. https://hackernoon.com/building-a-multiplayer-arkit-game-with-placenote-sdk-6bc736ee6760, 2018.

[MG+11] Peter Mell, Tim Grance, et al. The nist definition of cloud computing. 2011.

[Mim14] Onur Mimaroğlu. *Collaborative augmented reality*. PhD thesis, National University of Ireland Maynooth, 2014.

[MJ19] Martin Minárik and Adam Jurczyk. Multi-user interaction with 3d objects in augmented reality, 2019.

[MK94] Paul Milgram and Fumio Kishino. A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems*, 77(12):1321–1329, 1994.

[MSH+14] Héctor Martínez, Danai Skournetou, Jenni Hyppölä, Seppo Laukkanen, and Antti Heikkilä. Drivers and bottlenecks in the adoption of augmented reality applications. *Journal of Multimedia Theory and Applications*, 2:27–44, 03 2014.

[NS18] Michael Nebeling and Maximilian Speicher. The trouble with augmented reality/virtual reality authoring tools. In *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 333–337. IEEE, 2018.

[Pal18] Marco Paladini. 3 different types of ar explained: Marker-based, markerless & location. https://www.blippar.com/blog/2018/08/14/marker-based-markerless-or-location-based-ar-different-types-of-ar, 2018.

[Pap20] Chris Papenfuß. Ein hologram für alle (magie?). *Java aktuell – Das iJUG Magazin*, 03/2020, 2020.

[Per08] Simon Perry. Wikitude: Android app with augmented reality: Mind blowing. https://digital-lifestyles.info/2008/10/23/wikitude-android-app-with-augmented-reality-mind-blowing/, 2008.

[RC05] Jannick P Rolland and Ozan Cakmakci. The past, present, and future of head-mounted display designs. In *Optical Design and Testing II*, volume 5638, pages 368–377. International Society for Optics and Photonics, 2005.

[RKR19] Kimberly Ruth, Tadayoshi Kohno, and Franziska Roesner. Secure multi-user content sharing for augmented reality applications. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 141–158, 2019.

[RN95] Jun Rekimoto and Katashi Nagao. The world through the computer: Computer augmented interaction with real world environments. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*, pages 29–36, 1995.

[RSGC19] Xukan Ran, Carter Slocum, Maria Gorlatova, and Jiasi Chen. Sharear: Communication-efficient multi-user mobile augmented reality. In *Proceedings of the 18th ACM Workshop on Hot Topics in Networks*, pages 109–116, 2019.

[SFH+02] Dieter Schmalstieg, Anton Fuhrmann, Gerd Hesina, Zsolt Szalavári, L Miguel Encarnaçao, Michael Gervautz, and Werner Purgathofer. The studierstube augmented reality project. *Presence: Teleoperators & Virtual Environments*, 11(1):33–54, 2002.

[Sha18] Uri Shaked. Web-powered augmented reality: a hands-on tutorial. https://medium.com/@urish/web-powered-augmented-reality-a-hands-on-tutorial-9e6a882e323e, 2018.

[SHY+18] Maximilian Speicher, Brian D Hall, Ao Yu, Bowen Zhang, Haihua Zhang, Janet Nebeling, and Michael Nebeling. Xd-ar: challenges and opportunities in cross-device augmented reality application development. *Proceedings of the ACM on Human-Computer Interaction*, 2(EICS):1–24, 2018.

[SVdH13] Hanna Schraffenberger and Edwin Van der Heide. From coexistence to interaction: influences between the virtual and the real in augmented reality. 2013.

A Manual

A.1 XshARe Framework

A.1.1 Installation

Purchase Unity and download and install an up-to-date version. This project was developed and tested with version 2019.20f1. Later versions had compatibility problems with Xcode to compile a running iOS version. Get the provided XshARe project, load the included folder *Unity* as project in the Unity Hub and start it. Experience with Unity is required to work with XshARe. Experience with Mirror Networking and Microsoft Azure Spatial Anchors is advantageous. The project includes Mirror and Azure Spatial Anchors as extern components. In case of problems with those components, see the following links to look up where to get newer versions:

- **Mirror:** https://assetstore.unity.com/packages/tools/network/mirror-129321
- **Azure Spatial Anchors:** https://azure.microsoft.com/en-us/services/spatial-anchors/

You may want to create your own Microsoft Azure account and access your own installed web application to enable anchor sharing. Work through the following tutorials to get an understanding how to make Unity scenes working with Azure Spatial Anchors can be edited to access your own web applications url. All XshARe-related AR scenes use this feature.

- **Quickstart: Create a Unity Android app with Azure Spatial Anchors** - https://docs.microsoft
  us/azure/spatial-anchors/quickstarts/get-started-unity-android?tabs=azure-portal
- **Tutorial: Share spatial anchors across sessions and devices** - https://docs.microsoft.com/en-
  us/azure/spatial-anchors/tutorials/tutorial-share-anchors-across-devices?tabs=azure-portal%2CVS%2

A.1.2 Content of Asset Folders

Figure A.1 shows the included asset folders. Mentionable ones are:

- **AzureSpatialAnchors.Examples:** Includes four example scenes and the stat menu scene provided by Microsoft.
- **AzureSpatialAnchors.SDK:** The used SDK provided by Microsoft.

- **Mirror:** Includes provided Mirror components.
- **XshARe:** Includes developed elements constituting the XshARe framework
- **Mill:** Includes The mill application sample.
- **FurniturePlacement:** Includes the furniture placement application sample
- **CustomProject:** This is the most important folder. It provides a starting scene for developers to start a new project within. All other folders can be left untouched in best case.

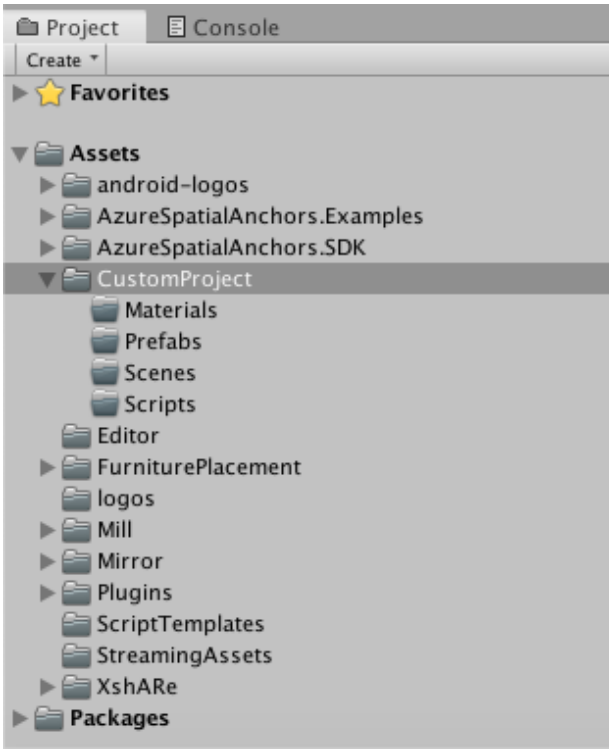

Figure A.1: The project's assets folder.

A.1.3 Compile and deploy a new Application

The compiling and deployment process is shown exemplary for an Android version. Go to the menu bar and click on *File->Build Settings...*. Inside the opening *Build Settings* menu, make sure, the *DemoLauncher*, provided by Microsoft is added as initial scene. All other additionally included scenes can be started from there, as long as their name begins with *AzureSpatialAnchors*. Add all AR scenes you want to run in the application. Take an AR-supporting Android smartphone and connect it to your computer via USB cable. It should now be available next to *Run Device*. Make sure you have developer mode enabled on your phone. Clicking on *Build and Run* creates a runnable .apk file on your computer and also tries to deploy it on your connected phone. If deployment was successful, the application called *HelloAR U3D* should start automatically. Figure A.2 shows working Build Settings as intended.

To compile a running iOS version of the Unity application, there are several further steps necessary. In Unity's *Build Settings* menu, select compilation for iOS. After clicking on *Build*, a

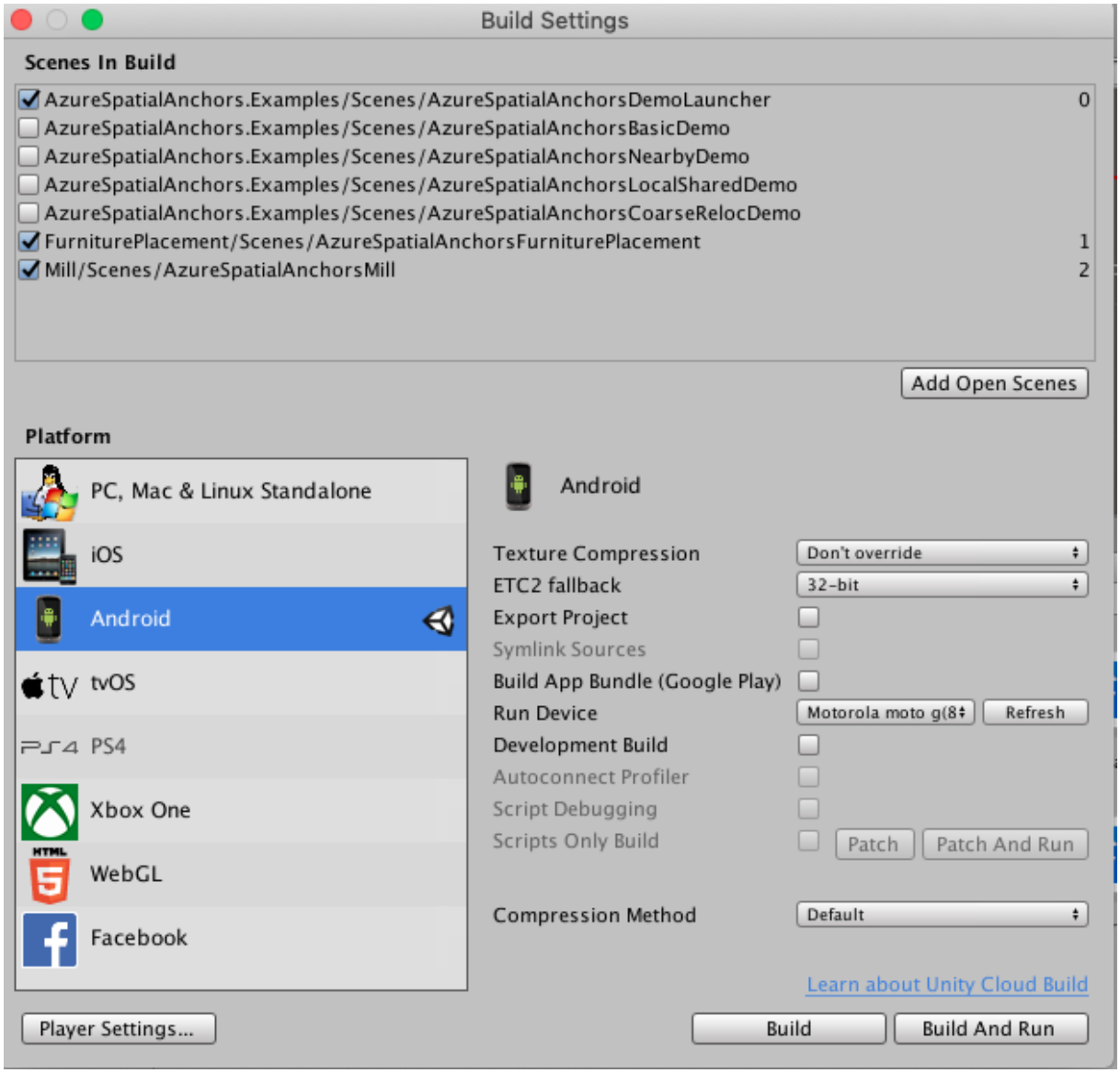

Figure A.2: The Build Settings dialog.

Xcode project is exported. Within this project, there is a file called *podfile*. This script needs to be loaded and executed in the application Cocoa Pods. The application enriches the application with further libraries necessary for compilation. In addition, a *.xworkspace* file is created, that can be opened in Xcode to combine the further libraries with the prior project. In Xcode, make sure to fix all possible problems on the *Signing and Capabilites* page of the project. Compiling and deployment to a connected iOS device should now be working.

A.1.4 Develop a new Application

The *CustomProject* folder is ready to be edited for a new multi-user AR application. It includes four child folders, as depicted in Figure A.1.

- **Materials:** Add new materials and images here.

- **Prefabs:** Add new prefabs here.
- **Scenes:** This folder includes the AR scene *SpatialAnchorsCustom*, which is intended to be your main scene to use. In addition, there is the normal 3D scene *CustomDesktopTest* to provide desktop testing.
- **Scripts:** Add new scripts here.

The *Prefabs* folder holds the *CustomPrefab* as an example of an intended GameObject. Several components are mandatory to make it work as synchronized GameObject in the AR scene, as depicted in Figure A.3.

- **Transform:** Position, Rotation and Scale
- **Mesh Filter and Mesh Renderer:** Required for visual GameObjects for visualization
- **Collider:** Required for interaction via raycast
- **Network Identity:** Mirror Component to synchronize GameObject
- **SyncDataContainer:** XshARe component to synchronize GameObject. Exchangeable with new MonoBehaviour

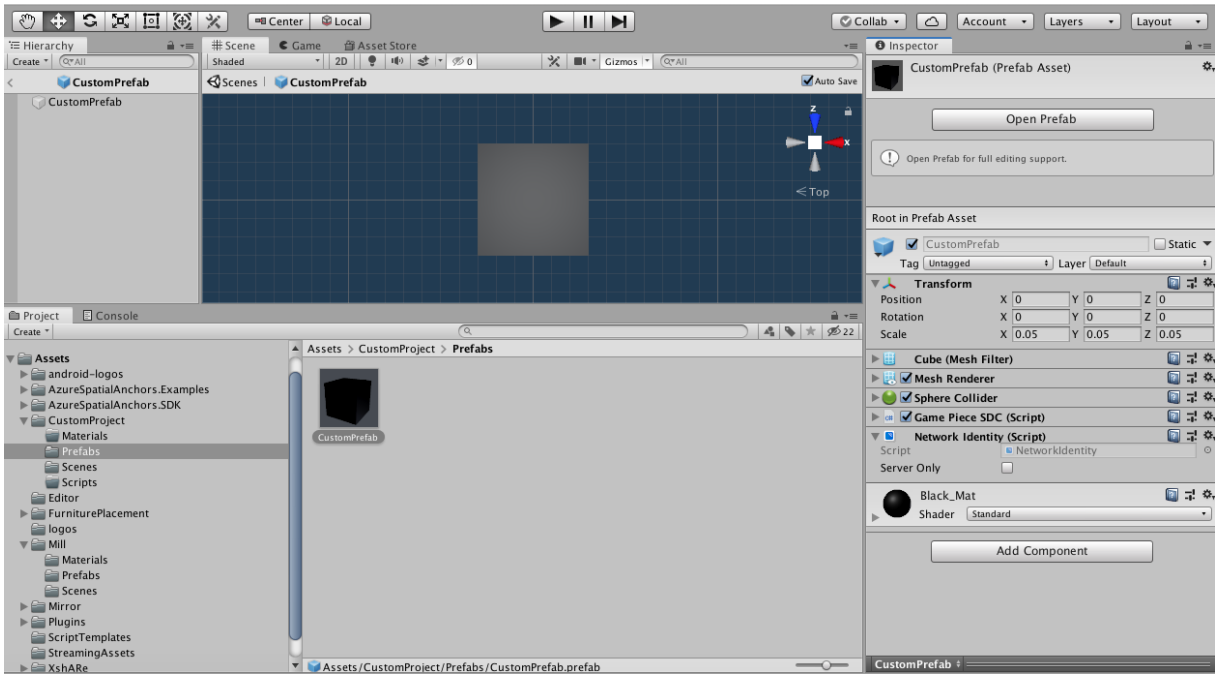

Figure A.3: The components of the CustomPrefab.

The folder *Scripts* already includes three rudimentary scripts as foundation to help the developer start coding.

- **CustomSDC** shows an example how to implement a new *SyncDataContainer* with a new synchronized integer field for metadata.
- **CustomModelLogic** can be used to implement specific business logic by overriding protected and public methods of *ModelLogic*.

- **CustomConfig** shows a simple proposal how to make custom GameObjects available for the *ModelLogic*.

Figure A.4 shows GameObjects initially included in the scene *AzureSpatialAnchorsCustom*. Two of them are intended to be edited by the developer.

- **Mirror:** This is where Mirror's NetworkManager component is attached. Every prefab intended to be synchronized needs to be added to the *Registered Spawnable Prefabs*. Mandatory are the prefabs *Room, MirrorServer and AnchorBasedRoot*.
- **Custom:** Attach your *ModelLogic* and *Config* here.

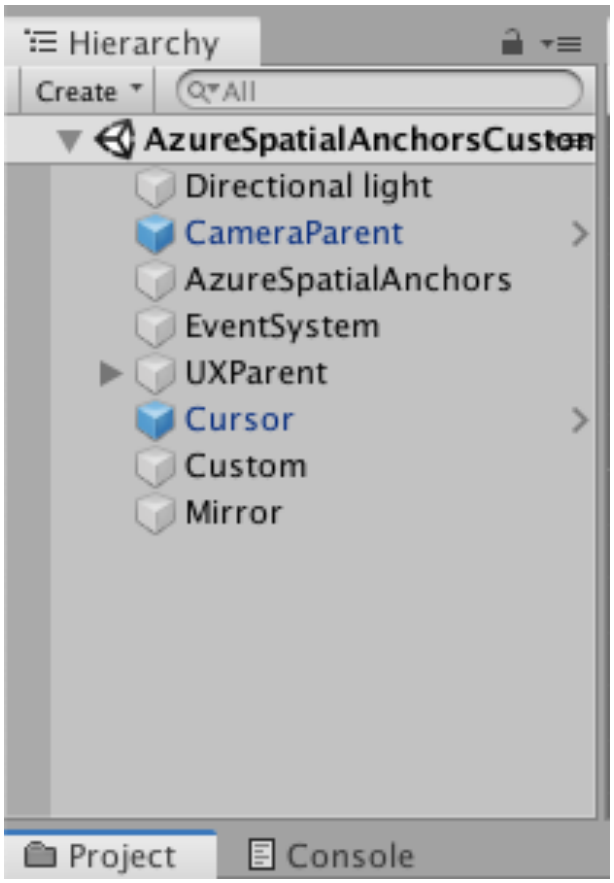

Figure A.4: GameObjects in the XshARe scene.

The scene is runnable on default and ready to get edited. Have a look in the furniture placement and mill folders to get further understanding how to use the project.

A.2 XshARe AR Applications

A.2.1 Standard AR Application Features

Figure A.5 recapitulates the standard preparation phase in 8 screenshots from the top left to the bottom right. Those are taken over from Microsoft's Azure Spatial Anchors Examples.

1. Connect as host or find a host as client to establish a network connection.

2. Start the workflow to create and share an anchor.
3. Tap the detected surface to set the anchor's position.
4. Move your device to capture more environment data.
5. Go through the workflow to locate the set anchor by all devices.
6. Start the AR session.
7. Touch the screen to interact with the AR scene.
8. Select virtual objects to manipulate their data.

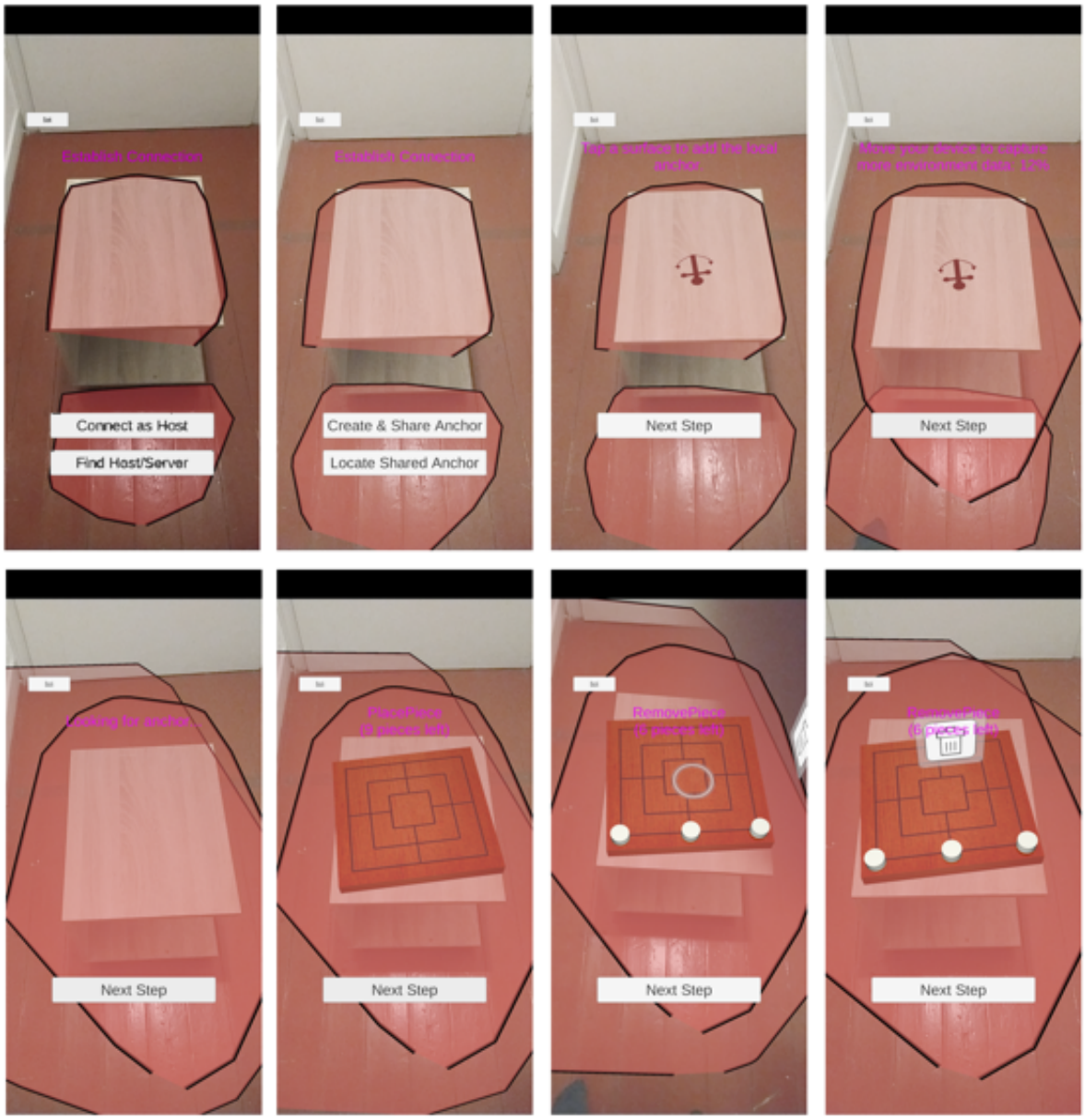

Figure A.5: Preparation and execution of the AR mill game.

A.2.2 Mill Game Application

Mill game interactions on handhelds, all according to the game rules:

- **Double tap:** Set new game piece
- **Single tap:** Select/Deselect game piece
- **Dragging the selected game piece to an empty field:** Move a selected game piece
- **Pressing the deletion button:** Remove a selected game piece

A.2.3 Furniture Placement Application

Furniture placement interactions:

- **Double tap:** Create a new furniture
- **Single tap:** Select/deselect a furniture
- **Dragging the selected furniture:** Move the furniture
- **Dragging next to the selected furniture:** Rotate the furniture
- **Pressing the deletion button:** Remove a selected furniture