Replicating augmented reality objects
for multi-user interaction
Master of Science Thesis
“There are only two industries
that refer to their customers as users.”

– Edward Tufte
Abstract

Augmented Reality (AR) is the combination of virtual objects and the physical world surrounding us. These virtual objects are used to enrich the real world. Because of technical improvements of mobile hardware, there are quite a number of AR applications deployed in the last decade.

To illustrate the potential of the AR technique and to look for a new concept of human-computer interaction, we started the 'Augmented Reality for 3D Multi-user Interaction' (ARMI) project. The goal of the ARMI project is to build an AR application where, two or more users, can work concurrently on the same virtual maquette. This maquette is visible through a Head Mounted Display (HMD) to display visual objects on top of the real world. The virtual maquette can be used for representing specific traffic situations with roads and cars but also for modelling other 3D scenarios. The following basic actions are supported: creating, selecting, moving, rotating, and deleting virtual objects.

The virtual maquette, build by the ARMI project, depends on four distinct areas and we identify the following research areas: hand tracking, hand-pose estimation, 3D interfacing, and AR-object replication. This thesis looks into the possibilities of object replication for the AR application.
First, a number of AR applications are considered and it becomes clear that most AR applications with multiple users are depending on a client-server approach and no object replication is used. Second, a number of replication systems are described in which more than one server is involved to keep the data consistent and a number of fundamental replication algorithms are discussed. Based on the related work the decision is made to use a speculative variant of an asynchronous majority consensus algorithm for the AR-object replication.

Furthermore, in this thesis the development, implementation and evaluation of the AR-object replication is described. From the evaluation it becomes clear that it is difficult to satisfy all the required replication parameters. We notice a number of replication limits. For example a scaling problem, which means that the number of clients is limited, and specific user behaviour in terms of performed operations per second. Based on the evaluation we conclude that the relaxation of a number of replication parameters is necessary to keep the system responsive enough for an AR application.
Acknowledgements

First of all I would like to thank my supervisor professor Marco Aiello from the University of Groningen whose suggestions, encouragement, and guidance helped me through the process of doing my master research and writing my thesis.

Especially, I would like to thank the members of the ‘Augmented Reality for 3D Multi-user Interaction’ project, Pieter Bruining for his support during the endless debug sessions, Gijs Boer for looking closely to the final version of the thesis for English grammar, and, finally, Maarten Fremouw for his support during the project.

I would also like to thank Hanneke Niessink for a personal conversation during my master research. This conversation was inspiring for me and important for making a personal decision.

Last, but certainly not least, I would like to give my special thanks to my wife Marianne for her constant support and love throughout the time I studied. Especially during my master research, which was a difficult period for both of us.

– Heino Lenting
List of Figures

1.1 An example of AR - A tag is replaced with a virtual teapot . . 4
1.2 A Vuzix iWear VR920 with camera . . . . . . . . . . . . . . . 7
1.3 Landmarkers are used to track motion and poses . . . . . . . . 8
1.4 Data flow of the entire system . . . . . . . . . . . . . . . . . . . 9
1.5 The simplified system architecture . . . . . . . . . . . . . . . . 10

3.1 Examples of Augmented reality (1) . . . . . . . . . . . . . . . . 20
3.2 Examples of Augmented reality (2) . . . . . . . . . . . . . . . . 21

4.1 Illustration of the 3D interface . . . . . . . . . . . . . . . . . . . . 30
4.2 Examples of a virtual world . . . . . . . . . . . . . . . . . . . . . 32

5.1 Overview of the AR-object replication model . . . . . . . . . . . 40
5.2 TCP and UDP is used for communication . . . . . . . . . . . . . 42
5.3 AR-object replication architecture . . . . . . . . . . . . . . . . . 43
5.4 A transaction’s life . . . . . . . . . . . . . . . . . . . . . . . . . . 48
5.5 The voting power is distributed across the replica managers . 51
5.6 Piggybacking is used to propagate state information . . . . . 54

6.1 An impression of the computer and 3D skills of the users . . . 73
6.2 A continuous task for the responsiveness experiment . . . . . . 78
6.3 Two different tasks are used to discover the AR user statistics 80
6.4 The average commit delay per replica manager configuration . 83
6.5 The maximum (above) and minimum commit delay . . . . . . . 84
6.6 The average network delay per replica manager configuration . 85
6.7 The maximum (above) and minimum network delay . . . . . . . 86
6.8 The update delay under different write resolutions . . . . . . . 87
6.9 The average update delay . . . . . . . . . . . . . . . . . . . . . . . 88
6.10 The feedback about the responsiveness per network delay . . 90
List of Figures

6.11 The responsiveness score per network delay . . . . . . . . . . . 91
6.12 The read/update operations per user for the first task . . . . . 93
6.13 The read/update operations per user pair for the second task . 94
6.14 The update resolution of user pair 1, 2, 3, 4, and 5 . . . . . . . 96
6.15 The number of operations by different users on the same object 97

A.1 The results of the performed operations on the cloud . . . . . 116
List of Tables

6.1 Software versions used for the experiments . . . . . . . . . . . . . . 68

B.1 The computer and 3D skills of the users . . . . . . . . . . . . . . . 118
# Contents

Abstract i

Acknowledgements iii

1 Introduction 3
   1.1 Augmented reality 3
   1.2 Application description 5
   1.3 The research areas 11
   1.4 Novelty 12
   1.5 Contribution 13
   1.6 Document structure 14

2 Replicating virtual objects for multiple users 15
   2.1 Problem definition 15

3 Related work 19
   3.1 Augmented reality 19
   3.2 Data replication 22
   3.3 Replication algorithms 25

4 Concept 29
   4.1 3D interface 29
   4.2 The virtual world 31
   4.3 Object replication 34
   4.4 Expectations 37

5 Realisation 39
   5.1 Introduction 39
   5.2 Communication 41
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3</td>
<td>System Architecture</td>
<td>43</td>
</tr>
<tr>
<td>5.3.1</td>
<td>Client</td>
<td>44</td>
</tr>
<tr>
<td>5.3.2</td>
<td>Replica manager</td>
<td>45</td>
</tr>
<tr>
<td>5.4</td>
<td>Transaction model</td>
<td>47</td>
</tr>
<tr>
<td>5.5</td>
<td>Majority consensus algorithm</td>
<td>50</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Voting process</td>
<td>50</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Speculative variant</td>
<td>52</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Progressive synchronization</td>
<td>53</td>
</tr>
<tr>
<td>5.5.4</td>
<td>Pseudo code</td>
<td>55</td>
</tr>
<tr>
<td>5.6</td>
<td>Cleaning the commit log</td>
<td>58</td>
</tr>
<tr>
<td>5.7</td>
<td>Minimizing the message size</td>
<td>59</td>
</tr>
<tr>
<td>5.8</td>
<td>Software development challenges</td>
<td>61</td>
</tr>
<tr>
<td>5.9</td>
<td>Known issues</td>
<td>63</td>
</tr>
<tr>
<td>5.9.1</td>
<td>Random transaction names</td>
<td>63</td>
</tr>
<tr>
<td>5.9.2</td>
<td>No key deletion supported</td>
<td>64</td>
</tr>
<tr>
<td>5.9.3</td>
<td>Maximum size of updates</td>
<td>65</td>
</tr>
<tr>
<td>5.9.4</td>
<td>No proxies possible without voting power</td>
<td>66</td>
</tr>
<tr>
<td>6</td>
<td>Experiment</td>
<td>67</td>
</tr>
<tr>
<td>6.1</td>
<td>Setup</td>
<td>68</td>
</tr>
<tr>
<td>6.1.1</td>
<td>Software and hardware</td>
<td>68</td>
</tr>
<tr>
<td>6.1.2</td>
<td>Logging system</td>
<td>69</td>
</tr>
<tr>
<td>6.1.3</td>
<td>Used write resolutions</td>
<td>70</td>
</tr>
<tr>
<td>6.1.4</td>
<td>Type of clients</td>
<td>71</td>
</tr>
<tr>
<td>6.1.5</td>
<td>Demographic data</td>
<td>72</td>
</tr>
<tr>
<td>6.2</td>
<td>Experiments</td>
<td>74</td>
</tr>
<tr>
<td>6.2.1</td>
<td>The commit delay experiment</td>
<td>74</td>
</tr>
<tr>
<td>6.2.2</td>
<td>The network delay experiment</td>
<td>75</td>
</tr>
<tr>
<td>6.2.3</td>
<td>The update delay experiment</td>
<td>76</td>
</tr>
<tr>
<td>6.2.4</td>
<td>Responsiveness of the system experiment</td>
<td>77</td>
</tr>
<tr>
<td>6.2.5</td>
<td>AR user statistics</td>
<td>79</td>
</tr>
<tr>
<td>6.3</td>
<td>Measurements</td>
<td>81</td>
</tr>
<tr>
<td>6.4</td>
<td>Results</td>
<td>82</td>
</tr>
<tr>
<td>6.4.1</td>
<td>The commit delay results</td>
<td>82</td>
</tr>
<tr>
<td>6.4.2</td>
<td>The network delay results</td>
<td>84</td>
</tr>
<tr>
<td>6.4.3</td>
<td>The update delay results</td>
<td>86</td>
</tr>
<tr>
<td>6.4.4</td>
<td>Results responsiveness of the system</td>
<td>89</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Augmented Reality (AR) is the combination of virtual objects and the physical world surrounding us. Virtual objects are used to add relevant information to the real world.

The development of mobile hardware makes the creation of real AR applications possible. Not only is improving the hardware but also all kinds of new techniques are available that can be used to support AR. Examples are the General Packet Radio Service (GPRS) and Google Maps. The field of AR is still in its early stages. In the last decade only a few initiatives are undertaken to use AR. Examples are virtual advertising in live TV streams, a mobile travel guide (Wikitude), but also games like ARQuake which is an AR variant of Quake.

Until now there was a technological barrier however today’s hardware, such as the iPhone 3GS [App], makes it possible to experiment with AR applications. From the current AR initiatives it becomes more and more clear that AR has a lot of potential. Therefore, the AR research area is interesting.

1.1 Augmented reality

For the sake of simplicity we assume that reality is what humans see. Therefore, every object that a human can see exists in reality. Besides reality there is also virtual reality (VR). With the VR technique the reality is completely replaced in a virtual one. There is no physical relation between
1.1. Augmented reality

reality and virtual reality. The user only sees the virtual world just like in 3D games. AR is a variation of VR. AR is based on the relation between reality and VR. With AR it is possible to show the user the reality and also add virtual objects to it. These virtual objects are presented in such a way that they fit in the physical world and look almost real.

An example of AR is given in Figure 1.1 where a virtual teapot is added on top of an existing table. The tag visible in Figure 1.1(a) is used for mapping the virtual teapot and the result is shown in Figure 1.1(b).

![Figure 1.1: An example of AR - A tag is replaced with a virtual teapot](image)

To give a user a realistic AR experience, a number of rules must be followed. First of all, accurate mapping of the virtual world against the real world is essential. This is a critical link and also one of the most difficult tasks to accomplish. To create this link between the real world and the virtual one, there are all kinds of localisation techniques available today. Examples are Global Positioning System (GPS) or localisation based on a WiFi energy map [VMOS05] which is a technique based on the signal strength of WiFi spots.

The problem with these techniques is that they cannot deliver the needed accuracy. Most of the localisation techniques are heavily influenced by buildings and the localisation inside buildings is quite poor. An accurate
1.1. Augmented reality

position is important to prevent that the virtual objects are constantly bouncing and to make sure that virtual objects have a fixed position. To prevent such mapping problems an accurate localisation technique must be used.

Another important aspect in improving the AR experience is to make the system intuitive. The human-computer interaction (HCI) becomes more and more important. Not only because the user is wearing the system but also because AR systems are always close by. Therefore, AR application must be intuitive. Techniques such as new input devices and easy to handle virtual objects are used to make the AR application intuitive. Traditional input devices, for example a keyboard and a mouse, are not suitable in a 3D context because these input devices are 2D orientated [Han97].

The final aspect in improving the AR experience is to have real-time response in visualisation and interaction. Because AR is the combination between reality and virtual reality, the virtual part of the AR environment must behave just like the physical objects in the real world. Everything from the new input devices to the replication of the virtual information must be done in (near) real-time.

What exactly is augmented reality? The AR definition given in [Azu97] says that a system is AR when it:

- Combines real and virtual,
- Displays 3D objects, and
- Is interactive in real-time.

In [Azu97] a complete overview of AR research is given. Numerous problems related to AR are described. Such as; how to deal with the localisation problem, dynamic errors such as system delays or lags, and the advantages and disadvantages of optical and video technologies for combining the real world with the virtual objects. Furthermore, the developments up to that point are summarized in [Azu97].
1.2 Application description

The goal of the AR project group is to create a completely working AR application. A prototype is built to show the possibilities of AR but also to get a better understanding of AR and what it means for the interaction between humans and computers. This application is called Augmented Reality for 3D Multi-user Interaction (ARMI).

Two or more users can work together on a virtual landscape that is presented in the reality. This landscape can represent a number of buildings or a specific traffic situation with roads and cars. The general idea is that users cannot only see the same landscape but also work together on the virtual objects. Examples of interactions are selecting an object, moving an object to another spot in the landscape, rotating an object, and deleting an object.

In each AR application several problems arise on how to display the virtual world and how to map the virtual world against the physical world. To make the maquette visible for the users a Vuzix iWear VR920 [Vuz09] is used as a Head Mounted Display (HMD). The Vuzix iWear are glasses with two small screens mounted in front of the eyes. This product is used for displaying virtual reality and therefore, it is not possible to look through the screens. For the project it is necessary that reality and virtual reality are combined into a single view for the users. Because the user can only see what is displayed on the screens the reality must also be presented on these screens. A Philips camera [Phi09] is mounted on the front of the Vuzix iWear. This camera is used to deliver a video stream to the screens inside the glasses to simulate see-through glasses.
1.2. Application description

In Figure 1.2 the Vuzix iWear is shown. The camera, used for delivering a video stream, is visible in front of the glasses, see Figure 1.2(a). From Figure 1.2(b) it becomes clear that one camera is placed between the eyes, which means that the user has no stereo vision of the physical world.

Using this setup the user can see the reality but also virtual objects can be added. David Drascic et al [DM96] have identified and described a number of issues in mixing the reality with virtual reality. The most relevant for this project are size and distance mismatches, limited depth resolution, and restricted field of view. It is crucial for a good AR application that the designers are aware of these problems.

The mapping of the virtual world against the physical world is done with visual landmarks. These marks are used by the visual marker detection system to track motion and to do pose estimation. The output from the marker detection system delivers the position of the display against the landmark. This position is used to render the virtual objects. Landmarks can deliver a high resolution for mapping the virtual objects against the real world. ARToolkit [WS03], ARTag [Fia05], and ARToolkitPlus [Kle07] are examples of landmark systems.
1.2. Application description

Figure 1.3: Landmarkers are used to track motion and poses

Two examples of landmark systems are shown in Figure 1.3. In the first example, Figure 1.3(a), ARTag is used and in the second example, Figure 1.3(b), ARToolkit is used for tracking motion and poses. More information about marker detection systems and a complete comparison about the available techniques can be found in [ZFN02].

The AR application ARMI can be divided into four components. The four components are: hand tracking, hand-pose estimation, 3D interfacing, and AR-object replication. These components together form the basis for the AR application. The components depend heavily on each other. The dependencies are shown in Figure 1.4.
1.2. Application description

In Figure 1.4 the data flow of the system is shown. An important role is played by the 3D interface, shown in yellow, which depends on all the other components. The hand-pose estimation is depending on the hand tracking and the AR-object replication is depending on the 3D interface.

To give an impression about the system architecture, a simplified version of the architecture is given in Figure 1.5. In Figure 1.5 four components are visible. At the top, the hand tracking component is visible. This component is responsible for tracking the hand in the video feed. The hand tracking component gives the hand-pose estimation 2 stereo images with the x and y coordinates of the hand together with the width and height of the tracked hand. The hand-pose estimation component, which is the second component, uses this information to match the tracked hand against the 3D poses and selects the most likely 3D pose.
1.2. Application description

Figure 1.5: The simplified system architecture
1.3. The research areas

The hand-pose estimation component sends the x, y, and z coordinates of the hand to the 3D interface component. The 3D interface component renders the virtual world and uses the coordinates from the hand-pose estimation to present the hand in the virtual environment. The 3D interface performs operations on the virtual world through the AR-object replication component. This component is located at the bottom of the system architecture shown in Figure 1.5.

From the AR definition it becomes clear that AR is about combining reality with 3D objects and interaction should be possible in (near) real-time. The reason for building a virtual landscape, as an example of AR, is because it contains all the elements needed for AR.

1.3 The research areas

In the previous paragraphs the four components used for the AR application are described and each component represents a research area. We identify the following research areas:

- Hand tracking,
- Hand-pose estimation,
- 3D interfacing, and
- AR-object replication.

The first area is hand tracking and is concerned with localizing a human hand in a video feed. A complete description about the used methods can be found in [Fre09]. After localization the image of the hand is used for hand-pose estimation which is the second research area and is described in [Boe09]. The hand-pose estimation is important for the AR application because the hand is used as the input device for the user.
Another research area is 3D interfacing. This area is about making the virtual environment practical and usable for multiple users. To make interactions in the virtual environment possible, a 3D interface is developed [Bru09]. The final area is AR-object replication which is necessary to keep the virtual world not only replicated but also consistent just like the real world.

The AR project group is formed by four master students of the University of Groningen following the ‘Software and Systems Engineering’ variant. The team members are: G. Boer (Hand-pose estimation), P. Bruining (3D interfacing), M. R. Fremouw (Hand tracking), and H. Lenting (Object replication).

In this thesis, the focus is on replicating AR objects for multi-user interactions which is important because there are concurrent users, located on different locations in the building, interacting in the virtual landscape.

1.4 Novelty

The novelty of this project lies in using commercial off-the-shelf (COTS) hardware to make an AR application work. This is possible because the technological barrier is moving. More and more AR applications are deployed in the last few months. Which means that (mobile) hardware is improving, the barrier is disappearing, and the COTS hardware is suitable for AR.

Most of the AR applications are based on single users and the users are interacting independent of each other. A step further is to design a concurrent AR application where multiple users are interacting. Concurrent users can work together, for example in a maquette. Therefore, the concurrent aspect of the system is quite interesting.

Building a prototype gives a good insight in the world of AR and illustrates the potential of the AR technique in many different areas. It is not only a lot of fun building an AR application, it is also a next step in building a totally new concept of human-computer interaction and therefore important.
1.5 Contribution

In this thesis the possibilities of AR are explored through the designing and building of a prototype called ARMI. ARMI is built using COTS hardware and uses an HMD with a built-in camera to display the virtual objects on top of the physical world. ARMI is a virtual maquette where multiple users can re-arrange and create virtual objects. The focus of this thesis lies on the replication of AR objects for multi-user interactions. To solve the replication problem a speculative variant of an asynchronous majority consensus algorithm is build. The virtual world is kept strong consistent to ensure that operations, performed by the users, are as realistic as possible. A cache located at the client and progressive synchronization is used to improve the responsiveness of the system.

Experiments are used to discover if the AR-object replication is a solution towards the problem statement. One of the results show a very specific user behaviour in terms of the number of update operations per second. The average number of operations per second for two concurrent users is ≈14. There are extreme spikes measured of more than 50 update operations per second. Which means that the number of operations per second, performed on the virtual world, is not constant and changes heavily. Another interesting fact from the results is that when more than 8 replica managers are involved it is difficult to distribute the voting power because of excessive UDP broadcasts. A message collision problem arises when the replica managers are sending their voting power on the same moment. This influences the commit speed heavily because a lot of syncing is required. From the results it is also clear that ARMI is scalable in terms of the distribution of operations to the clients and that the system can also handle the grow of read operations through the use of cache. Handling and committing the update operations is more difficult because of the fluctuating behaviour of the users.
Augmented reality is about combining real and virtual, display 3D objects, and is interactive in real-time. For the AR-object replication this last part, about (near) real-time interaction, is the most relevant and the most difficult part to accomplish. In this thesis it is shown that it is difficult to build such a replication system which can handle the update bursts of the users and at the same time keep it responsive. Therefore, the relaxation of a number of replication parameters is necessary.

1.6 Document structure

The remainder of this thesis is organized as follows.

The problem definition is given in Chapter 2 and the issues related to the problem statement are considered in detail. Current and prior research is given in Chapter 3. Relevant existing AR applications are described, a number of complete data replication solutions are discussed, and the most fundamental replication algorithms are given.

In Chapter 4 the concept is described. This chapter is used to motivate the chosen replication algorithm. The context of the AR-object replication is illustrated in this chapter and also a description about the 3D interface and the virtual world is given. The concept is used as basis for the realisation, which is in detail described in Chapter 5.

In Chapter 6 a number of experiments are defined and the results of these experiments are given. Furthermore, a discussion is included to review the results. Finally, in Chapter 7 a conclusion is given based on the results of the experiments and the future work is formulated.
Chapter 2

Replicating virtual objects for multiple users

Having multiple users interacting remotely on the same virtual world requires that virtual objects have to be replicated throughout the network. The central problem is replicating mutable object data in such a way that every user can read and write the data. A physical copy is placed on every client and replication must be used to synchronize the physical copies. Therefore research is done in the field of data replication.

Data replication is the process of maintaining a defined set of data at more than one location. It involves executing operations on the data and messaging with the other copy holders to synchronize the data [HNHM08].

2.1 Problem definition

In the context of ARMI the replication problem is defined as follows:

“The AR-object replication system must have the following features; strong consistency, high availability, scalability, tolerate copy and communication failures, no ownership or locking, minimize the size of the communication messages, synchronization must be fast, and finally, the system must be responsive for the user to work with.”

The problem statement represents three sub-problems. The first one is how to replicate virtual objects so that strong consistency is delivered by
the system. The second one is how to minimize the size of the messages sent through the network to reduce bandwidth consumption. The third sub-problem is concerned with improving the synchronization speed for the network.

Let us consider the issues of the problem statement in detail. The consistency of data is called strongly consistent if every read and write operation is serialized with respect to both read and write operations. Using this kind of consistency the system guarantees that every operation is globally serialized \[\text{FCC}^+\]. The term availability is used to define the accessibility of the object data. When a client wants to perform an operation on the data and the client cannot access the object data then the file is called unavailable. The object replication system must support high availability to ensure that the client can perform operations on the object data. Even when there are failures such as network partitions or node failures \[\text{TN}97\].

Scalability is the ability of the AR-object replication system to function correctly when the number of clients increases. The system must be able to handle a growing workload in a graceful manner. When the quantity of clients increases and the replication algorithm cannot handle the load, the system does not scale. The AR-object replication system must handle two types of failures. The system must tolerate communication failures and node failures such as disconnection failures. It is important to realize that failures have a great negative impact on the availability of the object data and because of that the handling of failures is important. Operations performed on an inconsistent copy are failures with respect to strong consistency however a temporary inconsistency itself is not a failure.

The object data that is replicated through the network does not work with (file)rights or ownership. Every client that is working with the AR application has the rights to perform create, delete, read, and write operations. There is no ownership of objects and thus every client has the same rights to perform the available operations. The AR-object replication must work without locking one or more copies. The operations are distributed as network messages and to reduce the bandwidth, the messages used for the operations must be as small as possible.
2.1. Problem definition

There are situations in which synchronization is needed to repair an outdated copy. A copy is outdated when the network has decided to process a given operation and the outdated copy does not have processed the same operation yet. For example, after a disconnect, in the case of a partitioned network, or simply a write initiated by one of the other copies. The synchronization of the copies must be as fast and efficient as possible to improve availability and to reduce the risk of a failing replica system. Because the object replication is used for an AR application the users expect fast and consistent responses to interactions. The interactions are performed through operations on the replicated object data and multiple users can collaborate to alter the data. When a user is performing an operation and a response is given without a noticeable delay the system is called responsive.

In the context of this work we assume the following. Every participating node in the replication process is connected through at least an 100 Mbps Ethernet network and the network must support Transmission Control Protocol (TCP) and User Datagram Protocol (UDP). The nodes must also be reachable through the network but it is possible that for a shorter or longer period of time there are network partitions or message delivery failures.
2.1. Problem definition
Chapter 3

Related work

AR looks like a good example of what Fred Brooks calls Intelligence Amplifications (IA) [Bro96]. IA is concerned with using the computer to perform or simplify a task. AR is used in quite a number of areas and becomes more and more popular.

3.1 Augmented reality

There are many more fields where AR is interesting to use. Other AR applications are military simulation systems like ModSAF [PGT99], see Figure 3.1(a), and playing Quake [ID] in an AR environment [PT00]. A game is an ideal prototype for HCI because of the entertaining nature of the interactions. Another advantage of using a game as a prototype is that limitation of the system can easily be hidden by the game play [SLSP00]. Another AR Quake variant is from Bruce H. Thomas [ATVP05] where the virtual monsters in the game are aware of the physical walls. An AR version of Mah-Jongg is described in [SEG98, Moba] and is illustrated in Figure 3.1(b).
3.1. Augmented reality

Virtual advertising is also a big AR area. Virtual advertising is used in sports broadcasting, see Figure 3.2(a). During a sport event, advertising is added to the live streams in real-time. Another sport related technique that is based on AR is the FoxTrax system [Cav99]. The FoxTrax system highlights the location of the, often difficult to see, hockey puck as it moves during the game. Wikitude is another example of AR and can be used as a mobile travel guide [Mobb]. Wikitude is build for the Android platform and uses the location based Wikipedia content. There are 350,000 points of interest worldwide and these points of interest are visible through Wikitude with the help of GPS. The content of Wikipedia is used to add relevant information to a specific GPS coordinate, see Figure 3.2(b).
3.1. Augmented reality

From the given AR applications it is clear that most of the AR applications can be divided into three types of AR applications. The first group are the AR applications where no users are directly interacting in the AR environment. The users are only viewing and no feedback is given to the system. The FoxTrax system and virtual advertising are good examples of AR applications with only viewing users. Such applications are data-distribution-wise not interesting because the data flow is one-way. On the other hand, this type of AR applications are interesting because of the render speed of the AR environment. The output is used for live TV streams which imply that rendering the 3D objects is done in real-time. This real-time aspect is possible because of the closed nature of such applications which means that there are no users interacting on the content visible in the 3D environment.

The second group of AR applications are the applications designed specifically for single user interactions. Training maintenance personnel and Wikitude are AR applications with single users. The 3D environment is dedicated to a single user and interaction in the virtual environment is only relevant for the user. No concurrent access to the 3D models is possible and these systems can run local on the user’s device. Which means that interactions can be performed locally and no external communication is needed. These types of AR applications are running independently and the Internet is often used to retrieve static information about the 3D

Figure 3.2: Examples of Augmented reality (2)
environment. This approach is interesting because the load is placed by the user himself and because everything is running locally, the system can be really fast. The downside is that the 3D environments are not concurrent. Most of the AR applications fit in this group for single users because it is the first logical step in building AR applications.

The last group of AR applications supports concurrent access to the virtual objects. Multiple users can view the AR environment and interact on the 3D objects. This type of AR applications are the most interesting applications for this project because in ARMI there are also concurrent users. Most of the AR applications with concurrent users are based on normal games. Popular to use is Quake and several AR variants of Quake are built. Another example is the AR variant of Mah-Jongg, which is also based on multiple players. The players of such an AR game are viewing the AR environment and the AR objects are controlled by a single server. Operations initiated by users are processed by the server and updates are distributed to the players. It is interesting to see that most of the concurrent AR applications are depending on a client-server approach. The problem with this approach is that it is not a solution towards the problem statement. A single server cannot have high availability because it is a single point of failure and the solution is also not scalable because the resources of a single machine are limited.

### 3.2 Data replication

So far in this chapter a number of AR applications are considered and often a single server is used to maintain the data used for the 3D environment. In the next few paragraphs a number of systems are described in which more than one server is involved to keep the data consistent.

The Gossip architecture is a replication approach where the data is replicated through replica managers that are near the users who are interested in the data [BGPS05, CDK05]. To provide a Gossip service two basic types of operations are available; query and update. Updates are propagated to other replicas and these replicas must distribute it further.
3.2. Data replication

The Gossip architecture supports causal ordering, which means that every recipient of update a and b receive update a before b. It also supports stronger ordering such as immediate ordering. Updates in an immediate ordering are applied in a consistent order with respect to any other update. Because of the nature of the gossip approach, updates are propagated to other replicas and are then distributed further, it is not likely that this system is responsive enough for ARMI.

The P2P Document Tree Management (DTM) is based on peer-to-peer (P2P) dynamic tree management in combination with hierarchical operation transformation over the document tree [SE98, PP07]. The data is represented as a tree and this hierarchical structure is used to lock at various levels. A user can make local changes to the locked data and cache these operations. When the changed section is shared the operations must be multi-cast to other users that are interested in the same part of the tree. Conflicts are resolved with domain specific rules. A problem of P2P DTM is that it is a solution towards a very specific hierarchical data structure and the data used for ARMI does not fit in this structure. Beside the fact that this approach cannot deliver strong consistency, locking is used by the user to perform operations. This locking is in contradiction with the problem definition for AR-object replication.

Woot (WithOut Operational Transformation) is a framework suitable for editing data in a P2P network. Woot has an optimistic approach towards operations performed on the data [OUMI06]. The Woot framework is based on two primitive operations. One for inserting data and one for deleting selected data. These operations must be monotonic. A unique character identifier and a sophisticated method are used to ensure consistency. Woot is not a solution towards the AR-object replication needed for ARMI because it has an optimistic approach. This optimistic approach means that the system does not expect conflicts and if there are conflicts they are fixed afterwards. The ARMI replication problem definition enforces a pessimistic approach and an optimistic system like Woot can never deliver strong consistency.
3.2. Data replication

The Mark & Retrace method retraces the documents address space to a valid state based on the time of the execution [GYZ05]. This is called Address Space Transformation (AST). After receiving an operation the document’s address space is retraced with the help of a technique called marking. After the transformation of the address space, the received operation can be performed. The Mark & Retrace method is designed for editing documents by multiple users. The execution order of the operations on the hosts can be different because of the different orders of arrival. The address space of the document is transformed to execute the operations. In the case of group editors this might be ideal but for ARMI this is not a solution. The reason for this is that in the case of ARMI each user is interested in the last version of the global document (virtual world) with one copy serialization of the operations.

The Bayou system is a replication technique where a number of replication managers across different hosts work together with a process called anti-entropy [TTPD95, CDK05]. With anti-entropy the updates are exchanged between two replicas. The replicas use each others write-log for comparison and merge towards a stable state. How to handle conflicts is left up to the user because the user knows best how to solve the problems. An unconnected user can update the data because a user can run its own replication manager. There are tentative and committed updates. Only the tentative updates can be used to get a consistent state. The ordering of the committed updates is decided by a primary replica manager. The Bayou system is not relevant for the AR-object replication because it also has an optimistic approach with the tentative updates and the committed updates. Another problem with this technique is that conflicts must be handled by the user which in the case of a 3D interface is quite difficult.

Deno is an object replication system specially designed for weakly-connected environments. Deno is based on weighted voting and uses pair-wise information exchange between replica managers [FCC+99]. In the Deno framework the clients are connected to replica managers and the replica managers are responsible for the consistency.
3.3. Replication algorithms

The communication between the replica managers is pair-wise to get a more flexible information flow. This is especially interesting in mobile, and thus often weakly-connected, environments. Deno is based on weighted voting and is a solution towards the given problem statement. The pair-wise communication is not useful in the context of ARMI because the participating users are not weakly connected.

A number of Internet companies are delivering cloud computing facilities. Cloud computing is a service which can be used for intensive calculations or other computer intensive tasks. Cloud computing is not only scalable, it also has a good availability and it is relatively cheap. Together with cloud computing usually a storage service is offered. Examples of data storage through cloud computing are Amazon SimpleDB and Google App Engine’s BigTable. In this case the replication of the data and the scalability is left to the cloud and the communication between the cloud and the client resembles a client-server architecture.

Cloud computing sounds ideal for ARMI but the speed of cloud computing is disappointing. During experiments performed on Google’s cloud, it appears that the round trip time of the cloud is fast, around the 25ms depending on your location. However, actually doing something with the cloud is another story. Reading costs at least 45ms and writing (which includes a read by the cloud) costs \( \approx 110 \text{ms} \). The complete experiment and its results can be found in Appendix A. The scalability and availability of cloud computing comes with a price in terms of high delays. Therefore, cloud computing is not suitable for ARMI.

3.3 Replication algorithms

From the given AR applications with concurrent users it is clear that a single server is used to maintain consistency. The client-server model is the most fundamental algorithm to keep consistency [CDK05]. It is also one of the most popular methods used to get one copy serialization which means that the operations are performed one at a time on a single copy. Most of the multi-player games are based on the client-server architecture and most of the known concurrent AR applications are also based on this approach. An example is the Quake 3 ‘first person shooter’ game. To support multiple
3.3. Replication algorithms

players across the Internet a Quake 3 server is used as a primary copy to keep the virtual game world consistent for the connected clients [Arm03a]. The client-server approach is simple and easy to maintain. The downside is that the solution is not scalable and that the availability is low since the server is a single point of failure.

The primary backup algorithm is based on a number of servers with only one primary. The clients can access the replicated data only through a primary. The primary is responsible for processing the updates and requests from the clients. A request is immediately processed and an update is propagated to the backup servers. When the primary server fails, one of the other backup servers takes over and becomes the new primary [BMST93, Cap90]. A failure of the primary server is not that straightforward to notice because backup servers are not aware if the server stops or if the primary is living in a network partition [Cap90]. The latter is problematic because when the network partition resolves there can be more than one primary and this may introduce inconsistencies in the system.

The token in the token algorithm is used to identify the primary from the backup servers. This algorithm is an extension on the primary backup algorithm. In the primary backup algorithm the primary never stops being primary unless it fails. However, in the token algorithm every backup server can become a primary. The selection of the primary is based on the exclusive token while the other backup servers have shared tokens [Cap90]. Servers with a shared token can handle read access while only the primary with the exclusive token can write. After a write the shared tokens are invalidated and new shared tokens must be generated. Losing the exclusive token due network or server failures is problematic and the token algorithm is vulnerable for the same problem as the primary backup algorithm, e.g. partitioning of the network.

The time stamp algorithm works based on the linear ordering of all events on a given server based on a logical clock. Every time an event happens the logical clock is increased with one. The ordering is per server and thus not a global ordering. To enforce global ordering of events every message sent between servers is accompanied by the logical clock of the sender. The receiving server must increase his own logical time until his own logical
3.3. Replication algorithms

clock is higher than the received logical clock. It is possible that events have the same logical time. When there is a conflict based on logical time the event from the lowest server id is processed as being the earliest. Based on the global ordering of events the consistency is maintained [Cap90].

The Coteries algorithm is based on the distribution of the master’s responsibilities to multiple servers. In the previous algorithms there is always one primary responsible for writing and that is problematic in the case of network partitions or server failures. The coteries algorithm works on a subset of participating servers [Cap90]. Every two of these subsets have at least one server in common. To perform a read operation every server inside a subset is checked for the highest version of the data. The data with the highest version in the subset is up to date and is returned. A write operation is performed on all the servers in a subset by locking them, performing the write operations, and increasing the version. Consistency is based on the requirement that every two transactions have at least one server in common based on the coteries. The given algorithm can handle network partitions and the system does not depend on just one primary server. Because the operations are performed in subsets the time it costs to distribute the operations depends heavily on the number of subsets which makes it in terms of speed less interesting.

The quorum algorithms are one of the most flexible replication algorithms. There are different variants of the quorum algorithms. Examples are Weighted voting, Read/Write quorums, and the Majority consensus algorithms [Gif79, AFN08, Cap90, RL03]. The principle is based on the distribution of voting power on the different servers. When a server agrees on a transaction, the server votes for the transaction. When the transaction has reached a certain voting power the operation is performed. The different variants work more or less the same. They differ in how the quorum is formulated to get a consensus. The voting algorithm can handle server failures and network partitions. The network partition with the majority of voting power will always continue. The quorum algorithms are flexible because when the parameters and the vote distribution are changed, most of the other replication algorithms can be described.
3.3. Replication algorithms

The Two-Phase commit algorithm is based on two phases: a voting phase and a decision phase. In the voting phase all the participating nodes are informed about a new transaction. All the nodes send a response to the initiating node about whether they agree or not. If all nodes agree the initiating node processes the transaction and informs every other node to commit the running transaction through a commit message. If at least one node does not agree on the transaction, the transaction is not executed and the nodes receive an abort message from the initiating node. The greatest disadvantage of this algorithm is that the nodes are blocked during the different phases until a commit or abort message arrives. There are situations possible that the algorithm will block indefinitely. One of those situations is an unreachable initiating node after the first phase. An interesting variant of the two-phase commit protocol specially designed for mobile wireless environments can be found in [NDDC05].
Chapter 4

Concept

The 3D interface depends heavily on the AR-object replication. Therefore, an introduction is given about the 3D interface and the virtual world to give an idea about the context where the AR replication system resides. More information about how the components relate to each other can be found in “Application description” (1.2).

4.1 3D interface

The 3D interface is a window to the virtual objects. Every user of the system has an interface and the users’s hands are used as an input device. The interface is specifically designed for users that are not familiar with 3D environments. Because there are multiple users in the system, the 3D interface must be concurrent. With concurrent is meant that more than one user can interact simultaneously with the virtual objects. Another important aspect of concurrency is that a user should be able to see what other users are doing in the virtual world.

The 3D interface is displayed on the Vuzix iWear VR920. Not only the interface is displayed, but also a live video stream of a camera mounted on the HMD is visible through the Vuzix iWear VR920. The combination of the physical world and the virtual interface is the basis for the AR part of the project and is illustrated in Figure 4.1(a). Every user is wearing an HMD and can interact with the virtual world through the 3D interface.
The interface supports the following tasks [Bru09]:

- Point out a virtual object to remote users,
- Move and rotate a virtual object,
- Add a new predefined virtual object to the virtual world, and
- Remove a virtual object from the virtual world.

To point out a virtual object in the interface the finger of the user is used, see Figure 4.1(b). More information about how the hand tracking is done and which technique is used for the hand-pose estimation can be found in [Fre09, Boe09]. When the finger enters an object the selected object becomes semi-transparent and every user can see this change to the object.

Figure 4.1: Illustration of the 3D interface

This technique, shown in Figure 4.1(b), can be used to point out a virtual object to remote users. When a user pinches an object a menu becomes visible. Through this menu the user can move, rotate and delete the object. New objects are created with the help of an Ipanel.
4.2. The virtual world

The Ipanel can be used to select one of the available objects by pressing the left or right arrow. The user can first select the new object and with the help of the move menu item decouple it from the Ipanel. After decoupling, the object is placed outside the Ipanel in the virtual world.

From the tasks described above it becomes clear that the 3D interface can execute operations on the virtual world. The AR-object replication is responsible for the execution of the operations initiated by the users. The 3D interface itself is not a part of this research. More detailed information about the 3D interface can be found in [Bru09].

4.2 The virtual world

The virtual world is a set of virtual objects and these virtual objects are used to represent the maquette. Multiple users are able to work in the same virtual world. The maquette contains several different virtual objects. The idea behind this maquette is that two or more users can work together to arrange the virtual world through the use of the 3D interface.

Landmarks are used for localization and are unique. These landmarks are essential to couple the virtual objects with the physical world. Each virtual object is related to one specific landmark. All the virtual objects that are visible in ARMI belong to the same virtual world. In Figure 4.2 two examples of the virtual world are visible to illustrate the possibilities.
4.2. The virtual world

Every virtual object has a basic data structure. The data structure is a dictionary. A dictionary is sometimes called an associative memory or an associative array. The index of the dictionary consists of an unordered set of key-value pairs. Because the dictionary is indexed based on the keys, the keys must be unique. The main reason for using a dictionary is that a simple copy-based approach can be used for transferring write operations. More information can be found in “Minimizing the message size” (5.7).

This dictionary is used to hold specific information for a given virtual object and is called a data definition. The data definition is used as a storage for key-value pairs. New key-value pairs can be easily added. Updating a value in the data definition is also straightforward.
4.2. The virtual world

A data definition in detail:

```python
datadefinition = { "filename" : "hand346576",  
                  "type" : "Hand.hand",  
                  "version" : -1,  
                  "scale" : 1,  
                  "position" : [0.0, 0.0, 0.0],  
                  "menuenabled" : False,  
                  "params" : {} }  
```

The given data definition is from a hand object. A hand object represents the physical hand of the user to other users in the system. Most of the elements of the data definition are easy to understand. Perhaps the key 'params' needs an explanation, the key-value pair is used at the moment of the object creation. There are situations in the 3D interface when, after the creation of an object, an event must be forwarded only to the user that initiated the creation. The 'params' key can hold such an event.

To get control of the virtual world, the virtual objects with their data definitions are placed in a data structure called objectlist. Every object living in the virtual world must be in the objectlist. The virtual world is nothing more than a set of data definitions of the virtual objects. The objectlist is a dictionary where the virtual objects are indexed based on the filename. This implies that every file name must be unique and that virtual objects with the same filename cannot exist, otherwise the existing data definition will be overwritten.

The data structure for the virtual world is given below. In this example only three virtual objects are visible.

```python
virtualworld = { "ipanel0000" : { [data definition] },  
                 "hand346576" : { [data definition] },  
                 "obje483564" : { [data definition] } }  
```

To keep the overview readable the complete data definition per object is omitted. The set of virtual objects is controlled by the AR-object replication to keep it consistent and available. The 3D interface expect four basic operations to change the virtual world.
4.3 Object replication

These operations are:

- Create a virtual object by adding the data definition to the objectlist,
- Update one or more key-value pairs in the data definition of an object,
- Read a complete data definition of an object, and
- Delete a virtual object by removing the data definition from the objectlist.

With the help of these operations the interface can perform all the tasks given “3D interface” (4.1). The problem definition in “Replicating virtual objects for multiple users” (2) reflects the requirements for the virtual world. To achieve this an AR-object replication system must be designed and built. How the AR-object replication controls the virtual world is described in the next sections.

4.3 Object replication

On the one hand, there is the 3D interface and on the other there is the virtual world. In the case of a maquette with only one user a local copy of the virtual world is the easiest and performance wise the most interesting solution because no replication is needed. The system must handle more than one user so a single local copy is not the answer. Therefore, AR-object replication is needed to control the virtual world.

The AR-object replication is responsible for executing operations on the different copies and is also responsible for keeping the copies consistent [HNHM08]. The operations performed by users are executed by their interfaces through the replication layer on the virtual world. The AR-object replication is the glue between the interface and a consistent virtual world. It now becomes clear that the AR-object replication has a great impact on the user experience of the interface. Fast and consistent replication is needed to ensure that the prototype is workable for the users.
4.3. Object replication

The biggest problem for the object replication is that the 3D interface is running at a maximum of 30 frames per second (FPS). Which means that the screen is updated roughly every 33 ms. To keep the data strongly consistent implies that every time an object is read (for drawing) the data must be verified. With verification is meant testing if the data is completely up to date. The verification process is handled by the replication logic. Local reads without verification are possible in the case of a Write All Algorithm however this algorithm has a number of negative side effects such as blocking when there are node failures [Cap90].

Imagine a virtual world with ten virtual objects. In that case the interface has to read ten different data definitions roughly every 33 ms. Note that there are multiple interfaces running, for each user one, so a huge load is placed on the replication layer. When looking at the normal Round Trip Time (RTT) between the users in a local network it is nearly impossible to achieve this kind of update rate. Another issue, closely related to this problem, is that the interface cannot wait for verification. The interface wants to redraw the screen as fast as possible which is in contradiction with the replication layer.

Another, more general problem, is the latency between users. Even in a local network the differences in the RTT between users is quite large. The ultimate goal of the AR program is to support multiple clients. Users with a high latency will have a negative influence on AR-object replication because for each replication algorithm communication is needed between the users to make a consensus or to get in a final state. If, for example, a user is located in the USA and the rest of the users are in Europe, the latency is more than 100 ms. If the USA user is playing a crucial role in the replication algorithm the replication will be quite slow. In an ideal situation a high latency must be a problem of the user that has the high latency and not for the entire object replication. It is even preferable that fast users can interact with the virtual world and slow users, in terms of latencies, are not allowed to interact at all to improve performance of the object replication for the rest of the users.

To solve the given problems above and to build a solution towards the problem statement an asynchronous quorum algorithm is chosen with a
simple and effective cache. Asynchronous protocols have a number of advantages over synchronous protocols. Such protocols can work even when the network is not completely connected, it can easily adapt to changes in the group membership, and asynchronous algorithms have fewer demands on the underlying network [KC99, FCC+99].

The reason for using a quorum algorithm is because the availability is high and the algorithm can handle copy and communication failures and it delivers strong consistency. Every copy in a quorum algorithm has a certain voting power. To execute a transaction a consensus must be reached. That makes the algorithm quite flexible because the voting power can be distributed in different ways over the participating copies. Often a uniform voting power distribution is used to maximize the availability and to improve the performance. The voting power can also be placed on one single copy which simulates a primary backup approach. The voting power distribution can be used to make a balance between availability and performance [KC99] which makes it very flexible. The quorum algorithm ensures mutual exclusion among conflicting operations and no locks are needed to work towards a final state.

There are different variants of the quorum algorithm such as Weighted voting, Read/Write quorums, and the Majority consensus algorithms [Gif79, AFN08, Cap90, RL03]. An implementation based on the Majority consensus algorithm is chosen for ARMI. With a majority based algorithm only the majority of the copies needs to agree on an operation. More information about this algorithm can be found in the “Realisation” (5).

The cache mentioned before is used to obtain maximum read performance for the 3D interface. The use of the cache, has of course, impact on the strong consistency of the system. When a user is reading from a cache the data can be stale. Together the replica managers are responsible for keeping the virtual world consistent. A client can propose an update to a replica manager after which the replica manager is responsible for the replication of the update. After updating their local copy the replica managers are sending the updates to the cache located near the client. Which means that the virtual world is kept consistent and that the operations performed on the cache of the clients are sequentially correct.
4.4. Expectations

It is not guaranteed that the client reads the most up to date data. A client can also directly contact a replica manager to perform a read operation that is guaranteed to be the last version but this method is never used by the interface.

The users are not directly involved in the replication process and they have no replication logic running. This means that the clients are light-weight. This is good because rendering the interface and the rest of ARMI is quite heavy on the computer. Furthermore, with this approach the latency towards a client does not have an impact on the replication process because the client itself is not participating in the voting process.

4.4 Expectations

The AR-object replication is in between the 3D interface and the virtual world. Most of the replication expectations are driven by the interface. The interface is completely dependent on the replication layer and to ensure correct functioning of the interface the replication layer must function as expected.

Not only the basic operations must be supported by the AR-object replication system but the replication layer must also be a solution towards the given problem statement to keep the virtual world consistent, scalable, and available. Applications based on video rendering have a very distinct read/write ratio because the system is reading most of the time and fast reading is possible through the cache.

A quorum based algorithm can handle the problems given in the problem definition and the virtual world can be kept consistent. Adding a cache to the clients prevents that the clients are constantly reading from the AR-object replication system. The price is that the data can be stale. The time that the data is outdated on a replica manager can be further reduced by a fast synchronization method.

It is expected that the majority consensus algorithm in combination with a cache, placed at the client, fulfills the needs for the 3D interface and the rest of ARMI.
4.4. Expectations
Chapter 5

Realisation

In this chapter the implementation of the concept is described. The realisation is based on the concept and detailed information about the AR replication is given in the following sections.

5.1 Introduction

The AR-object replication system is part of a bigger project and the goal is to develop an AR application called ARMI to illustrate the possibilities of AR. There are different components in ARMI and the AR-object replication is such a component. These components are very diverse and finding a suitable programming language is not easy. For hand-pose estimation and hand tracking a programming language is needed that is really fast. For interfacing and object replication a more flexible programming language is needed.

The programming language C is the most efficient and thus the most suitable programming language for building CPU intensive components. The problem with C is that it is not flexible and developing a program in C can be difficult. Almost the complete opposite is Python. Python is an object oriented programming language and comes with extensive standard libraries. Python supports smooth integration with other programming languages. For the AR application both languages are used. Performance orientated components are based on C and components which must be more adjustable are written in Python. The best of two worlds are combined into
5.1. Introduction

one AR application. The AR-object replication is to be built completely in Python. Speed is essential for the AR-object replication but the replication speed does not depend on the CPU but on network latencies. A faster CPU does not automatically increase the replication speed because the delay comes from the network. Another reason for using Python for the replication system is because the 3D interface is constantly evolving. Therefore, the replication layer must be easy to adjust.

The project is developed on Linux/Unix and Mac OS X and the final AR application will run on both systems. The components written in Python can run on both platforms. Most of the components written in C can easily be ported because Mac OS X is also Unix-based just like Linux. There are a few differences between the platforms but the porting itself is not a problem. Windows is not supported because the project group works only with Linux/Unix and Mac OS X. Porting the application to Windows is feasible.

Figure 5.1: Overview of the AR-object replication model
5.2 Communication

An overview of the system is given in Figure 5.1. The AR replication system contains clients and replica managers. Each replica manager maintains a copy of the virtual world and operations performed on the local copy of the replica manager are sent to the connected clients. Each client receives the operations and performs them on the local cache. Detailed information about both the client and the replica manager is given in “System Architecture” (5.3).

5.2 Communication

Before delving into details about the clients and the replica managers an overview of the communication protocols used in the system is given. The clients have no replication logic and depend on the replica managers. It is important that a reliable data transmission is used to ensure sequential consistent data. For the communication between the client and the replica manager TCP is used. TCP delivers the following reliability guarantees; retransmission when a message is not received, buffering is used to balance the stream between sender and receiver, checksums are used to ensure the correctness of a message and, in this case very important, sequencing [CDK05]. Sequencing means that a TCP stream is a FIFO protocol. The receiver receives the information in the same order as the sender has sent it.

The replica managers also communicate with each other. Of course, TCP can be used for communication between replica managers but TCP is connection oriented which means that before a connection is made, both parties must cooperate in the establishment of the connection. This can be problematic when there are multiple replica managers and data must be sent to all replica managers. This is not possible at once with TCP. Sending data to all replica managers at once is important because each outgoing message is always addressed to all other replica managers. Because of that UDP is a better choice. From UDP it is know that it is not connection oriented. This means that you can send a message to all replica managers (broadcast) or send it to a group of replica managers (multicast) or to just one replica manager (unicast).
UDP cannot give the same guarantees as TCP [CDK05]. Delivery is not guaranteed and messages can be duplicated or can arrive unordered for example. Applications based on UDP must handle these issues themselves. The AR-object replication layer can handle the UDP characteristics because of the chosen replication technique and because Message-Digest algorithm 5 (MD5) is used to validate the incoming UDP messages for corruption.

The TCP protocol and UDP protocol are both present on the transport layer in the Open Systems Interconnection (OSI) protocol model. The transport layer is the lowest level at which messages are handled instead of packets [CDK05]. Figure 5.2 illustrates the usage of the TCP and UDP in the system.
5.3 System Architecture

The AR-object replication model is illustrated in Figure 5.1. This model represents the basic architecture of the AR-object replication. The different copies of the virtual world used in ARMI are visible. Every 3D interface has a built-in client and one or more clients connect to a replica manager. This connection is based on TCP and each replica manager can handle a number of clients. The most important task of the replica managers is to work together towards a consensus and the second task is to feed the clients with updates.

Figure 5.3: AR-object replication architecture
5.3. System Architecture

The system architecture is shown in Figure 5.3. The client and the replica manager are shown and both have a replicated copy of the virtual world. At this point it is important to realize that these copies do not have the same consistency properties. The data located at the replica managers is guaranteed up to date when a read is performed but the data located at the client can be stale.

5.3.1 Client

The client part in the AR-object replication architecture has three components that are related to the replication logic. The fourth component, displayed in gray, illustrates the 3D interface. The 3D interface uses the Datastore for reading data however the 3D interface cannot write to the Datastore directly. Write operations initiated by the users are forwarded by the 3D interface to the Data controller for further processing at the Replica manager. At a given point these operations are returned and performed on the Datastore.

The client has the following components:

- The TCP client component is used to connect to a replica manager. The complete virtual world is sent by the replica manager when the client connects for the first time. The TCP client is ready for sending and receiving operations after connecting and receiving the virtual world.

- The Data controller maintains the data located at the Datastore. Every operation performed on the data is done through the Data controller. The operations are received from the Replica manager through the TCP client. The Data controller can initiate a new operation on the data by sending it to the Replica manager.

- The Datastore holds the virtual world.

These components together form a light-weight client and can easily be used by the 3D interface. The client is completely built according to the Object-Oriented Programming (OOP) method. The virtual objects in the virtual world are independent objects and are living in the 3D interface. The operations initiated by the users are performed directly on these objects. The
5.3. System Architecture

objects handles the read operations themselves and other operations are forwarded to the Data controller. This means that the 3D interface works only with a number of objects and is not aware of the different operation types and how to handle these operations.

5.3.2 Replica manager

The Replica manager has eight components in total. The Replica manager contains the following components:

- A TCP server is used to manage the connections with the clients. The TCP server is responsible for keeping the connection alive and to register the clients at the Client manager. Proposed operations from the client are received by the TCP server and send directly to the Consistency controller.

- The Client manager is used to manage the clients. When a new client is registered by the Client manager the data from the Storage manager is sent to the client with the help of the TCP Server. After committing a transaction the Client manager is notified. At that moment the Client Manager sends the same operations to the connected clients.

- The UDP server is used to receive UDP messages from other Replica managers. The UDP server is responsible for checking the integrity of the messages based on an MD5 checksum and is responsible for the forwarding of the messages to the Consistency controller.

- Sending a message to all reachable replica managers can be done through a single call to the UDP client. The UDP client uses multi-cast to send messages to the other Replica managers. Before sending the message an MD5 checksum is created and added to the network message. The receiving UDP server can validate the message based on the checksum. There are no delivery guarantees given by the UDP client.

- The Storage manager is used as a storage for the virtual world and the consistency controller uses the Storage manager for performing operations on the virtual world. The Storage manager is also available for the Client manager. The Client manager can read the virtual world to push a complete copy to the new clients. The data is stored in memory.
The Transaction manager is responsible for storing all transactions. The transactions are used to hold the operations and is described in more detail in “Transaction model” (5.4). The Consistency controller uses the Transaction manager for inserting transactions and for storing votes from the other replica managers.

The Commit log is used as backup for outdated Replica managers. When a specific transaction is not completely processed by all Replica managers a copy of the transaction is placed in the Commit log. The transaction can be used for synchronization when the outdated Replica manager asks for it.

The most important component of the Replica manager is the Consistency controller. Messages for connected clients and participating replica managers are processed by the Consistency controller. Every message is placed in a FIFO queue and a state machine approach is used to process the messages one by one. Before an operation is performed on the Storage manager a consensus must be reached based on input from other replica managers. To do this the Consistency controller uses the UDP server, the UDP client, and the Transaction manager.

The Consistency controller plays the central role in the replication system. A state machine approach is used so there are no locks necessary in the system and because of that no deadlock can occur.

Most of the given components in the Replica Manager are relatively easy to build. The replication logic is concentrated in the Consistency controller and the rest of the components are only helpers for the Consistency controller. This approach is chosen to make the system more understandable and more maintainable.

In the Client, the virtual world is represented by OOP objects. The Replica manager is not working with OOP objects works on a set of object definitions which does not have any meaning for the Replica manager at all. Operations such as reading an object are, in the case of the Replica manager, nothing more than reading a plain object definition.
5.4 Transaction model

Transactions are used to perform a sequence of operations on the AR-object replication so that they do not interfere with other operations performed by other concurrent clients. The transactions must be completely successful or completely aborted.

The ACID properties are formulated by Haerder and Reuter. The properties describe the features of a transaction. ACID stands for: Atomicity – a transaction is either performed completely or not at all, Consistency – a transaction is used to bring a copy from one state to another consistent state, Isolated – a concurrent transaction must be processed sequentially, and Durable – after committing a transaction the new state must be permanent [HR83, CDK05].

The transactions are used to perform operations on the virtual world. These operations are initiated by the 3D interface. The 3D interface has a very distinct behaviour. Transactions always contain single operations because the 3D interface does not support multiple operations in one transaction. A new location is proposed to the replica manager when the 3D interface wants to move an object to a new location. The new location is based on the client’s last known location of that object. The operation which is sent to the replica manager also contains the last version known to the client. Based on this information a decision can be made by the replica manager about processing or aborting the transaction.

The transaction model is illustrated in Figure 5.4. In this model the client and the replica manager are visible. In a nutshell, a transaction is triggered by the client and the result is received back from the replica manager. The yellow boxes in Figure 5.4 are decision points in the transaction process. Detailed information about the ‘Voting phase’ can be found in the section “Majority consensus algorithm” (5.5).
Figure 5.4: A transaction’s life
5.4. Transaction model

Each operation is initiated by a user wanting to change the virtual world. The user can interact with the virtual world through the 3D interface. The 3D interface is allowed to read directly from the cache to improve read performance. This is the first decision made in the transaction model and is named 'Local read?'. Read operations can be handled directly and the other operations are forwarded to the replica manager.

When an operation arrives at the replica manager the first thing that the replica manager does is checking if the operation is conflicting. This decision is called 'Is conflicting?'. An operation is called conflicting when the version number is lower than the version stored in the local copy. In other words; the client initiates an operation on stale data. When this is the case the operation is aborted and otherwise a transaction is created.

The voting phase starts after creation of the transaction. In this phase the replication algorithm is used to make sure that concurrent transactions are processed sequentially and to make sure that each replication manager commits and aborts the same transactions. This phase is described in “Majority consensus algorithm” (5.5). The result of the 'Voting phase' can be used for the next decision point which is called 'Majority?'. From a given transaction it is known whether the transaction has enough votes to get the majority of votes. Based on the outcome, the transaction is either aborted or committed. After committing the transaction is placed in the commit log. More information about 'Add to commit log' can be found in “Cleaning the commit log” (5.6).

The next step is to inform each client that is connected to the replica manager. Finally, each client receives the transaction initiated by one client. This operation is then performed by the client on the local cache. So, in the replica manager the transaction is first committed in the consistent virtual world and then the clients are asked to do the same on their local copies of the virtual world.

The transaction model given in Figure 5.4 nicely illustrates the differences in operations. Read operations are performed directly by the client without using the replication logic. Other operations use the replica manager to proceed.
5.5 Majority consensus algorithm

Transactions are used to perform operations on the virtual world. The transactions must be distributed across the participating replica managers and processed on each copy of the virtual world.

From the ACID definition it becomes clear that transactions must be atomic, consistent, isolated, and durable. Which means that a replication algorithm is needed to make sure that the concurrent transactions are processed sequentially. After processing the new state becomes consistent. A transaction must be completely committed or completely aborted, and to ensure durability the new state must be available next time.

5.5.1 Voting process

Eventually the replica managers must find consensus about whether a given transaction must be committed or aborted. This is needed to keep the virtual world consistent and to make sure that each copy of the virtual world is in the same state. Using network communication the replica managers are working together to perform the operations initiated by the 3D interface.

This can be done by voting. Voting in this context is for the first time described by Gifford in [Gif79]. Gifford designed and implemented a file replication system based on assigning voting power to each copy of the files. Each copy has his own voting power and uses this voting power for voting on transactions. The main algorithm described in [Gif79] is now known as a Read/Write quorum algorithm. In such a system different quorums can be used for read and write operations. To perform a read operation an R quorum is collected and for a write operation a W quorum is needed. By changing the R and W quorum it is possible to change the characteristics of the system in terms of performance or reliability.
5.5. Majority consensus algorithm

Figure 5.5: The voting power is distributed across the replica managers

The same principles are used by the Majority consensus algorithm. With one difference; read and write operations use the same quorum. The used quorum is \( R = W = \left\lfloor \frac{n}{2} \right\rfloor + 1 \) where \( n \) is the number of participating replica managers. This quorum is known as the majority [Cap90]. Each transaction needs to collect a majority before it can be committed. The distribution of the voting power is illustrated in Figure 5.5. In this example three replica managers are visible and the total voting power of the system is exactly 1.0.

Recall from “Transaction model” (5.4) that transactions are used to combine related operations on the virtual world. Before a transaction can be committed on a replication manager a quorum needs to be collected. A transaction can only be committed when the majority of the replica managers agree on this transaction. Each replica manager can receive or create new transactions. Every time a new transaction is received, a vote is given. A vote is a message containing the sending replica manager’s id, the unique id of the transaction, the voting power given by the sender, and the sender’s logical clock. The logical clock is a local counter that is incremented each time the replica manager performs an event such as sending a vote message [Lam78]. Each replica manager collects vote messages from other replica managers. During the voting process more and more votes are given for running transactions and finally some transactions receive a majority with respect to others.
5.5. Majority consensus algorithm

To get strong consistency each transaction must be committed in the same order at all replica managers by providing mutual exclusion on all the running transactions. This is done by using the logical clock of the votes given per transaction. Every replica manager has a set of votes given for a certain transaction. Because each time the logical clock is increased this implies that in the vote set there is always a lowest logical clock per replica manager. Such a vote is called a top vote and is the first known vote from a specific replica manager. In an ideal world with no concurrent transactions and \( n \) replica managers there are \( n \) top votes given per transaction. Because concurrent transactions can exist, see “Speculative variant” (5.5.2), a transaction is committed when a transaction has a majority according to the other top transactions. After committing a top transaction, existing conflicting transactions must be aborted because these transactions should never be committed because they are not relevant anymore.

5.5.2 Speculative variant

When a new transaction arrives or a new transaction is made at a replica manager the decision is made if this new transaction is a candidate or not. In other words; does the transaction have potential according to the other running transactions. A new transaction has no potential when there is already voted for another conflicting transaction. In this case the new transaction is blocked. Such a blocked transaction can only get potential when the conflicting transactions are aborted. This is a conservative approach and is often used in pessimistic distributed protocols.

An optimistic alternative is to vote for each transaction even when there are conflicting transactions. Using an optimistic approach the transactions, even the conflicting ones, collect votes and thus make progress in the vote process. Which transactions must be aborted or committed is left up to the network. This is based on the logical clocks and the commit rule. In the set of conflicting transactions there can only be one top transaction with a majority. So each replica manager votes for each transaction and after collecting the votes the decision can be made about which transactions must be executed on the virtual world.
5.5. Majority consensus algorithm

This speculative approach is useful for the AR-object replication because progress is made thus the replication speed improves. The downside of this approach is that more messages are sent, even messages containing transactions that do not have any potential at all. More information about speculative voting and the results of experiments with an optimistic and pessimistic approach can be found in \cite{FCC009}.

5.5.3 Progressive synchronization

Synchronization is used to repair outdated copies of the virtual world. An important role in the synchronization process is performed by the commit log. An outdated replication manager can be repaired by using the backup transactions located in the commit logs of other replica managers.

In replication algorithms based on quorums the replica managers can learn from new transactions with a majority that they are self outdated. This behaviour is illustrated by the following example. When top transaction $T_c$ with version 3 has enough votes but the version in the local copy is 1 the replica manager knows there is gap between the transactions. Top transaction $T_c$ must wait until the replica manager is up to date. The replica manager initiates a sync request and sends it to the other replica managers. The current version of the data is placed in this sync request. This version is important because other replication managers can search through their own committed transactions in the commit log to find the missing transaction. This transaction $T_b$ with version 2 is send back to the outdated replica manager in a sync reply. The replica manager can repair itself with transaction $T_b$ and continue with committing $T_c$. At that moment the replica manager is completely in sync and functions normally.

In this traditional synchronization method the outdated replica manager must first learn that other replica managers are ahead in the replication process. In worst case this means that a new transaction must have a majority before a gap is notified by the replica manager. This situation can be slightly improved by a more progressive synchronization style. Progressive synchronization is possible when a replica manager knows the state of the other replica managers. This is problematic because the replica managers are not aware of the state of the other replica managers. One way
of learning more about the other replica managers uses a technique that is called piggybacking. With piggybacking information is added to every message sent by the replica managers. Relevant information in this case is the last version of the locally stored data.

With the help of piggybacking other replica managers can learn that a specific replica manager is outdated. This is based on the stored version added to each message. A replication manager that receives a message with a lower version number than the local version number can immediately send a sync reply. When a replication manager has a higher version number it means that a transaction with a majority is committed and that the replication manager is running ahead of the sender of the message. With progressive synchronization other replica managers can help fixing outdated replica managers.

Figure 5.6: Piggybacking is used to propagate state information

The idea of using piggybacking is illustrated in Figure 5.6. There are three replica managers. Two replica managers are running version 78 and one replica manager is slightly behind with version 77. The other replica managers can learn from the piggybacked information that the replica manager, represented in yellow, is outdated. A sync request is sent even before the replica manager itself knows it is out of sync.
5.5. Majority consensus algorithm

Fixing an outdated replica manager is slightly improved based on some knowledge about the other replica managers. Knowledge about other replica managers also plays an important role in cleaning the commit log. Detailed information about this subject can be found in “Cleaning the commit log” (5.6).

5.5.4 Pseudo code

The following pseudo code is a representation of the code running at the replica managers. The voting phase with the speculative variant and the progressive synchronization is shown in the pseudo code. The pseudo code gives an impression about the voting phase rather than a complete overview of the algorithm.

The main while in the pseudo code works as a state machine. The program never escapes from this loop except when the replica manager is stopped. The UDP and TCP messages are handled by the same state machine and the messages are placed in MSGQUEUE and processed one by one. Each time a message arrives at the replica manager the condition is set. This is important because when there are no messages the state machine is waiting on this condition.

The pseudo code starts with three global variables. The first one is a FIFO queue named MSGQUEUE and each message from the network arrives in this queue. The second variable is a transaction list. The running transactions are placed in ACTIVETRANS. The last global variable, COMMITTRANS, is used as a commit log and contains the committed transactions.

MSGQUEUE = queue of received UDP/TCP messages
ACTIVETRANS = list of running transactions
COMMITTRANS = list of committed transactions

while True:
    if MSGQUEUE length is 0:
        wait for condition
while MSGQUEUE length is > 0:
    MSG = MSGQUEUE.pop(0)

if MSG from self:
    continue

if MSG is SYNCREPLY:
    if MSG.transaction is valid:
        send vote for MSG.transaction to other
        commit MSG.transaction
        clean up ACTIVETRANS
        insert MSG.transaction in COMMITTRANS
        update connected clients

    for each TRANS in ACTIVETRANS with majority:
        commit TRANS
        clean up ACTIVETRANS
        insert TRANS in COMMITTRANS
        update connected clients
    continue

if MSG is SYNCREQUEST or sender is behind:
    if TRANS for sync in COMMITTRANS:
        send TRANS as SYNCREPLY
    continue

if MSG is TRANSACTION and valid:
    if MSG.transaction is unknown:
        insert MSG.transaction in ACTIVETRANS
        store local vote in MSG.transaction
        send local vote for MSG.transaction to other

    if ACTIVETRANS contains MSG.transaction:
        store remote vote for MSG.transaction
5.5. Majority consensus algorithm

```
clean up commit log

for each TRANS in ACTIVETRANS with majority:
    commit TRANS
    clean up ACTIVETRANS
    insert TRANS in COMMITTRANS
    update connected clients

continue

if MSG is OPERATION:
    if MSG is valid:
        insert MSG.transaction in ACTIVETRANS
        store local vote in MSG.transaction
        send local vote for MSG.transaction to other

continue
```

A number of abbreviations are used in the pseudo code to keep the code readable. Each message arriving at the replica manager is of the type SYNCREPLY, SYNCREQUEST, TRANSACTION, or OPERATION. A message of the type SYNCREPLY is used to repair an outdated replica manager and a SYNCREQUEST message is used to request a SYNCREPLY. A message can also be of the type TRANSACTION or OPERATION. TRANSACTION means that the message comes from another replica manager and OPERATION means that the message comes from a connected client.

The given pseudo code illustrates the basics of a replica manager. The code can be divided into four distinct areas with different tasks. There is one part concerned with receiving operations from the connected clients. For these operations a transaction is built and distributed across the network. Another part is used for collecting votes from other replica managers. The last two parts are related to repairing an outdated replica manager.
5.6 Cleaning the commit log

When a vote is given for a transaction this vote is sent through UDP to all participating replica managers. This message also contains the transaction itself. The distribution of transactions and votes are combined into one message. This is possible because of the relatively small size of the operations.

Transactions that are processed and committed by the Consistency controller are placed in a log located in the Commit log component. A copy of the transactions are placed in the log as a backup. This backup is used when, at a certain moment, another replica manager asks for a specific transaction. This situation occurs when another replica manager is out of sync. In other words, the commit log is used as backup to fix out dated replica managers.

The problem with commit logs is that they can grow indefinitely. In the AR-object replication an algorithm based on voting is used and the individual replica managers are not aware whether other replica managers are up to date or not. The replica manager only knows when a majority of the replica managers has voted for a specific transaction. Because there is no guaranteed knowledge about the other replica managers the commit log grows forever. In an ideal world the commit log is completely empty because all the replica managers are in sync. However, this is a utopia because the replica managers are never completely up to date at the same time. The only exception is when the system is settled and the replication layer is not used by the interface. Garbage collect the items of the commit log is important but in many replication research not much time is spent on how to do this efficiently.

The solution for keeping the commit log as small as possible is based on knowledge about the participating replica managers. A transaction is committed when the given transaction has enough votes from other replica managers. Afterwards it is placed in the commit log. The committed transaction in the commit log thus only contains the voting power of replica managers that agreed on this transaction. From “Majority consensus algorithm” (5.5) it is clear that the total voting power of the system is always 1.0. Which means that if the voting power given for a transaction in the commit log is 1.0 then every replica manager has voted for that specific transaction and so, it can be removed from the log. It is not practical that replica managers
must communicate about whether a specific transaction can be removed from the commit log or not. Therefore, the same piggybacking technique is used for the synchronization as described in “Progressive synchronization” (5.5.3).

The stored version of an object definition is piggybacked on every message sent by a replica manager. The receiving replica manager can put the voting power of the sending replica manager on each transaction in the commit log which has the same or a lower version number. Because the stored version is used, it is guaranteed that the sending replica manager is up to date until that version. So no backup is needed for this replica manager until the given version. Because each replica manager gives their latest version number finally all the voting power of the committed transactions becomes 1.0. Cleaning the commit log is now easy; remove each transaction with a voting power of 1.0 from the commit log. This method for cleaning the commit log implies that it can happen that an outdated replica manager, without voting power, cannot be repaired by other replica managers. More about this problem and how to solve this issue can be found in “No proxies possible without voting power” (5.9.4).

With piggybacking the replica managers communicate about their current state. This information is used to clean the commit log. During the failure of a replica manager the commit log grows until the failing replica manager is up to date. When the failing replica manager is up to date the commit log is reduced to only a few running transactions. This behaviour is exactly the idea behind the commit log.

## 5.7 Minimizing the message size

The AR-object replication system supports four types of operations. Each operation is sent through the network as a network message. The size of the message depends on the type of operation and the size of the update. A message for reading and deleting is quite small. However, a message for creating or writing an object has all the initial object data enclosed in the message which makes is relatively large in size. Compression techniques are widely used to make data transfer more efficient and to reduce the size of storage. In [LH87], a large number of data compression methods are described. Rsync is perhaps the most used method for synchronizing copies.
5.7. Minimizing the message size

on different host because it is a very elegant way of synchronizing based on
rolling checksums and it is efficient [TM96, Lan01]. Other methods are the
copy-based approach and the Lempel Ziv Welch (LZW) algorithm [SM02].

To work towards a solution it is important to realize that reducing the
message size involves two sub-problems. The first problem is a file synchro-
nization problem and the second problem is a delta compression problem
[SM02, KpV02, Tri99, TTM+02]. A message is sent from one host to
another across a communication link. The receiving host is unaware of
the message body. At that moment a file synchronization problem arises.
How to transport the message to the receiving host while minimizing the
communication costs. The message for the write operation can be further
reduced with the help of delta compression. Instead of sending a completely
new data definition for each time a write is performed, only the differences
are sent between the current version of the object definition and the new
object definition. This is possible because the sending host and the receiving
host have the same reference data.

The operations are performed on the virtual world. The virtual world has
a number of virtual objects and each virtual object has its own collection
of variables. Such a collection is called a data definition. As said before
in “The virtual world” (4.2), the data definition of a virtual object is based
on a dictionary. This is done because a difference between the old and the
new version can be easily extracted. For example, consider the following
simplified object definitions:

```python
datadefinition_old = { 
    "filename" : "hand346576",
    "type" : "Hand.hand",
    "version" : 5,
    "position" : [3, 5, 1] }

datadefinition_new = { 
    "filename" : "hand346576",
    "type" : "Hand.hand",
    "version" : 6,
    "position" : [8, 2, 5] }
```
5.8. Software development challenges

Both the sending and receiving host have the datadefinition<sub>old</sub> and can therefore be used as a reference for the delta encoding. The sending host first builds a new datadefinition<sub>new</sub> and then compares this data definition with the data definition<sub>old</sub>. They differ only at the 'version' key and the 'position' key:

\[
\text{delta} = \{ \\
\quad "version" : 6, \\
\quad "position" : [8, 2, 5] \\
\} 
\]

Because both parties involved in this problem have the same reference file, the datadefinition<sub>new</sub> can be easily constructed by the receiving node with the help of the delta. This solution is just like the copy-based approach but in this case the keys are used for splitting the data definition into pieces. The solution is not that low level as most of the delta compression methods but it is a straightforward solution to reduce the size of the write operations.

Before sending the message to the network they can be further compressed with GNU zip. This looks promising but all the messages are smaller than the maximum size of a UDP packet so compressing the messages is not always needed. Compressing and decompressing each message is CPU intensive and because the messages are very small it is questionable if this improves the overall network speed and bandwidth usage. Therefore, in the current prototype only the simple delta compression is used to reduce the message size and compressing the messages with GNU zip is offered as an option.

### 5.8 Software development challenges

During the development of ARMI many software development issues and challenges arise. Most of them are low-level programming issues and related specifically to ARMI and are therefore not very interesting. There are also issues and challenges that are worth telling because they illustrate the problems on a higher level and can be useful in other AR situations.

The first and most difficult challenge was the fact that ARMI depends on four different areas; hand tracking, hand-pose estimation, 3D interfacing, and AR-object replication. These areas must work harmoniously with each other which could be difficult. In some situations the areas are even conflicting. A good example is the relationship of the 3D interface and the
5.8. Software development challenges

AR-object replication. The 3D interface is only interested in rendering the 3D virtual objects as fast as possible to get a smooth and fast interface. The AR-object replication is concerned about the consistency of the virtual objects. These aims are contradicting because the interface wants to read objects without any delay and the object replication wants to make sure the data is up to date which means a delay by definition. To make an application work, where so many different disciplines are combined together, it is very important for each member to focus on the final application. Good communication is needed between the different areas to build the final application.

Another problem is the commit storm. A commit storm is a large number of commits and is a product of the interactions in ARMI. When a user performs an action in ARMI, for example dragging an object, two objects are involved. The virtual hand of the user and the virtual object are constantly changed according to the input device. The virtual hand and the virtual object are not changing at the same time but at different moments in the render phase so the updates are not combined. Also, changing the direction and the angle of a virtual object is not performed at the same time. This introduces a commit storm with many small updates. This illustrates the importance to think distributed. The development of a distributed system is not comparable with a stand-alone application.

The 3D interface is used to display the virtual world and the user can perform interactions on the objects in the virtual world. This means that each operation is initiated by a user based on the current state of the object. Recall from the introduction in “Realisation” (5) that a local copy of the virtual world is placed at the client for fast reading. This cache implies that it is possible that the data in the local copy is stale. Therefore, it is possible that a user initiates an action based on outdated information. Imagine two concurrent users viewing the same virtual object. The first user moves the virtual object 10 inches to the right. The local copy of the second user is not completely up to date and the client also performs the move to the right. For the last user the situation is difficult to understand because after performing the operation, the object is moved two times (20 inches) to the right when the user is finally up to date.
5.9. Known issues

To prevent such cases it is important that replica managers verify that clients perform operations on objects based on the last known version of the data. This means that only operations can be performed on the virtual world when the user is viewing the most up to date version.

The 3D models visible in ARMI are very diverse in size. Some models are only 20 KB others are over one MB. The 3D interface does not support editing of virtual objects which means that the model data is static. Therefore, the model data is not replicated. Each client has its own models on disk and if needed the client can read the model data directly. No replication logic is needed. It is a good choice to put static data outside the replicated virtual world because the data is static which means no changes are allowed. This keeps the replicated data as small as possible to improve performance.

5.9 Known issues

The AR-object replication built for ARMI is a prototype because the application is only used to explore the possibilities. There are known issues that may introduce problems when the prototype is used in production environment. The most interesting issues are explained and a solution is given to reduce the risk or to improve the situation.

5.9.1 Random transaction names

A transaction name is used to identify a transaction uniquely. It is critical for the system that the generated transaction name for new transactions is unique. Duplicated transaction names for different transactions will introduce vote errors. In other words, the replica manager can vote for another transaction then intended.

A transaction name is based on two parts. The first part is a prefix for a given object type to make it more readable for humans. The second part is a random number with a fixed length. The combination of the prefix and the random part makes the transaction name uniquely enough for the prototype.
5.9. Known issues

The solution to make sure that the transaction is really unique is to use the host name and the port of the initiating replica manager. The combination of host, port, prefix, and a random part makes the transaction unique for that replica manager. In this case it is important that a new transaction name is tested against the locally running transactions and the transaction in the local commit log. When the generated transaction name does not exist in both lists, the transaction name is unique for the complete network.

5.9.2 No key deletion supported

A data definition is used to store object specific information and is based on a dictionary. The elements in such a data definition can easily be accessed through a key that is used as an index in the dictionary. Writing to the dictionary is easy and finding a difference between a new and old version is straightforward. More about this can be found in “Minimizing the message size” (5.7).

The 3D interface is used by the users to interact with the data definitions of the virtual objects. A virtual object can be created, deleted, updated and read by a user through the interface. Create, delete, and read are operations performed on the complete data definition. Write is possible on a key-value pair in the data definition itself. A key-value pair from a data definition is never deleted by the 3D interface. When a given key is used for a virtual object this key will stay in the data definition until the complete data definition is removed. Key deletion is not supported by the AR-object replication. In the case of the 3D interface this is not a problem because there is no need for removal of the key-value pairs.
5.9. Known issues

To support the deletion of key-value pairs a small adjustment is needed in the AR-object replication. The AR-object replication already supports writing to a key-value pair. Changing the value is based on the key and it is possible to change the value with an empty string. In the current version this means that the given key with an empty string as value is still located in the data definition. To support key deletion the elements in the data definition must be tested after performing an update. The test is nothing more then testing per key-value pair for an empty string. Each key-value pair with an empty string as value can be deleted. Because each copy of the virtual world performs the same test after an update, the data is kept consistent.

5.9.3 Maximum size of updates

An update has currently a maximum size of a UDP packet minus the overhead. The maximum size of a UDP packet is 65535 bytes based on the fact that the length field in a UDP packet is 16 bit. The exact size of the packet is much smaller because it depends on the maximum packet size of the underlying protocol, for example Ethernet. Most of the software and network hardware use hard limits on the UDP packet size and there is not a standard. To prevent problems the maximum size of a UDP packet for the prototype is set to 1500 bytes. Updates bigger than 1500 bytes will be ignored and no response will ever arrive from other replica managers.

A solution is to break the big messages into smaller chunks and send them separately. The receiving process must understand that the data can be fragmented and is responsible for collecting the different chunks and combine them into one message. UDP does not give any guarantees about whether a given message will be delivered and in which order. Therefore, numbering of the chunks is not enough. The receiving processes must also know how many chunks they must receive before the message is complete. Another solution is to break the updates itself into smaller updates and push them independent of each other to the replica manager as different transactions. In that case the updates are not bounded so there is no guarantee that all the transactions will be executed.
5.9.4 No proxies possible without voting power

Proxies can be used to improve availability of the system by representing unavailable replica managers in the network. The voting power from the failing replica manager is moved to the proxy. Furthermore, a planned disconnection can be handled by a proxy [Kel99]. Proxies can also improve the commit latency because the voting power of the system is distributed across only a few replica managers near the users and the other replica managers function as proxies [KC99].

Using proxies to represent failing or disconnected replica managers is possible in the AR-object replication. Using proxies for concentrating voting power on only a few replica managers is problematic. When there are replica managers without any voting power in the network it is not guaranteed that when such a replica manager is outdated it can be repaired based on other commit logs. Recall from “Cleaning the commit log” (5.6), a committed transaction is removed from the commit log when a given transaction has a voting power of 1.0. In other words; every replica manager must have processed the transaction and no backup is needed. When there are replica managers in the network without voting power, it is possible that the commit logs are empty but there are still replica managers (thus without voting power) that are not in sync.

If there are replica managers needed without any influence on the voting process there are two solutions. The first one of course is to give the proxies a very small amount of voting power. Such a replica manager does not play an important role in the vote process but the commit logs are not cleaned when the proxy is outdated. When a proxy is out of sync it can be repaired using the commit logs of the other replica managers. The second solution is to keep the voting power 0.0 per proxy and use another synchronization technique. An outdated proxy without voting power must connect as a client on another up to date replica manager to get an up to date copy. It is important to realize that the last solution influences the consistency of the proxy. There can be a gap between the last transaction processed by the proxy and the rest of the network. Because of that the first solution is preferred.
Chapter 6

Experiment

A number of experiments are defined to evaluate ARMI. The experiments are used to get a better understanding about the AR-object replication itself and how it performs together with the 3D interface.

The following five experiments are used to evaluate the AR-object replication:

– Commit delay, the time used by the system to collect a majority of the votes and to commit a transaction
– Network delay, the time it costs to initiate an operation by a client and commit it to the replica manager and send a reply to the client
– Update delay, the time it costs to update all connected clients
– Responsiveness, indication of the users about whether the AR-object replication is still responsive
– AR user statistics, discover the users behaviour with respect to the number of operations performed on the virtual world

The first three experiments focus on the middleware. These experiments are used to give an impression about the AR-object replication speed and the scalability of the system. The other two experiments are based on the user experiences. From these results it becomes clear how the users behave in a virtual world and how fast the system should react to keep it responsive. The discussion about the AR-object replication is based on the results of all the experiments.
6.1 Setup

In this section the setup and the conditions used for the five experiments are described in detail.

6.1.1 Software and hardware

ARMI depends on a number of software libraries. In Table 6.1 the software and their respective versions are given.

<table>
<thead>
<tr>
<th>Software</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARMI</td>
<td>0.2</td>
</tr>
<tr>
<td>Python</td>
<td>2.5.2</td>
</tr>
<tr>
<td>ARToolKit</td>
<td>2.72.1</td>
</tr>
<tr>
<td>Numpy</td>
<td>1.2.1</td>
</tr>
<tr>
<td>PyOpenGL</td>
<td>3.0.0b8</td>
</tr>
</tbody>
</table>

Table 6.1: Software versions used for the experiments

The experiments can be divided into two groups. The first group is concerned with the middleware and the second group focuses on the user experience. For both groups a different hardware setup is used and this hardware setup is given per experiment.

- Dell XPS M1530 (laptop)
  The operating system is Ubuntu 8.04 (Hardy Heron) running on an Intel Core Duo CPU T7250 at 2.00GHz with 2GB memory, a Marvell Technology Group Ltd. 88E8040 PCI-E Fast Ethernet Controller and an nVidia Corporation GeForce 8600M GT video card.

- MacBook Pro (laptop)
  The operating system is Ubuntu 8.10 (Intrepid Ibex) running on an Intel Core Duo CPU 2.16 GHz with 3 GB memory, a Marvell Yukon Gigabit Adapter 88E8053 Singleport Copper SA Ethernet controller and an ATI Radeon X1600 Mobile video card.

- HP Compaq (desktop)
  The operating system is Debian 5.0 running on an Intel Core Duo CPU
6.1. Setup

E6550 at 2.33GHz with 2GB memory, an Intel Corporation 82566DM-2
Gigabit Ethernet controller and an nVidia Corporation GeForce 8400
GS video card.

Eleven HP Compaq desktops, one Dell laptop, and one MacBook Pro are
used for the experiments.

6.1.2 Logging system

Logging is used to keep track of what the users are doing during the experi-
ments. This is important because a log file can give detailed information
about the behaviour of the users. Using this log file it is pretty easy to look
back to the experiments and extract specific actions or operations performed
by the users. This helps to get a better understanding of the experiment
itself. The logging system is only used for the experiments where users are
involved.

All the operations performed on the virtual world such as create, write,
read, and delete are logged. These operations are initiated by actions, for
example clicking on an object, performed by a user. The operations in the
log files are chronological correct because the operations are distributed to
the commit log through TCP and are only stored after finding a consensus
by the replica managers. The actions are distributed directly to the commit
log which means that actions are not stored globally chronological correct
but are only correct per client. This means that the log file is, in the case of
operations, chronological correct and that the time between the operations
is known. Small fluctuations are possible because the RTT between the
logging system and the replica manager is not always constant.

The logging system is based on a special prepared replica client. This
client functions as a normal replica client but the difference is that this
client records all the information that it receives from a replica man-
ager. This information contains the operations performed by the users
of ARMI. In a normal setup the actions, performed by the users, are not
distributed across the network but in the case of the logging system the
users send notifications with the performed action to the recording client.
 Afterwards, the operations and the actions are combined into a single log file.
The logging system uses Pickle, which is an object serialization method of Python. The operations and actions are objects and the serialized representations of these objects are stored in the log file. This means that the log file is not human readable. The next few lines illustrate the data that is stored as an example.

<table>
<thead>
<tr>
<th>objectname</th>
<th>action</th>
<th>version</th>
<th>id</th>
<th>deltetime</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>hand575385283934</td>
<td>create</td>
<td>-1</td>
<td>0_2</td>
<td>5.10617685318</td>
<td>[data]</td>
</tr>
<tr>
<td>hand575385283934</td>
<td>update</td>
<td>0</td>
<td>0_2</td>
<td>0.02393293381</td>
<td>[data]</td>
</tr>
<tr>
<td>obje354765248585</td>
<td>create</td>
<td>-1</td>
<td>0_2</td>
<td>0.12899899482</td>
<td>[data]</td>
</tr>
<tr>
<td>obje354765248585</td>
<td>update</td>
<td>0</td>
<td>0_2</td>
<td>0.02865409852</td>
<td>[data]</td>
</tr>
<tr>
<td>hand575385283934</td>
<td>update</td>
<td>1</td>
<td>0_2</td>
<td>0.02743792533</td>
<td>[data]</td>
</tr>
<tr>
<td>hand575385283934</td>
<td>update</td>
<td>2</td>
<td>0_2</td>
<td>0.03995609284</td>
<td>[data]</td>
</tr>
<tr>
<td>hand575385283934</td>
<td>update</td>
<td>3</td>
<td>0_2</td>
<td>0.03093194961</td>
<td>[data]</td>
</tr>
<tr>
<td>hand575385283934</td>
<td>update</td>
<td>4</td>
<td>0_2</td>
<td>0.06035304070</td>
<td>[data]</td>
</tr>
<tr>
<td>obje354765248585</td>
<td>update</td>
<td>1</td>
<td>0_2</td>
<td>3.55972599983</td>
<td>[data]</td>
</tr>
<tr>
<td>obje354765248585</td>
<td>update</td>
<td>2</td>
<td>0_2</td>
<td>0.03758096695</td>
<td>[data]</td>
</tr>
</tbody>
</table>

The following values shown are: objectname, type of action, version of the object, id of the initiating client, and the delta time in seconds. The data itself is left out to keep it readable. The logging system is used to record the experiment where users are involved and it shows a unique view of the experiments. The results can be used to look to specific actions and behaviour afterwards.

6.1.3 Used write resolutions

For a number of experiments a write resolution is used. The write resolution is the number of write operations performed on the virtual world in a given period. The values used are: 25, 22.2, 20, 18.2, and 16.6 operations per second. This means that the time between operations for the five write resolutions are: 40ms, 45ms, 50ms, 55ms, and 60ms.
6.1. Setup

The chosen values are based on a number of tests used to find the write resolutions where problems arise. It is important to realize that if the AR-object replication can handle a write resolution of 40ms the AR-object replication can also handle the other write resolutions. The reason for this is that the time between the operations grows per write resolution. Which means that there is even more time to find a consensus.

6.1.4 Type of clients

Three types of ARMI clients are used for the experiments. Every experiment where users are involved is based on ARMI clients with a complete 3D interface. The users can use this interface to interact on the virtual world.

In each experiment where a write resolution is used, a specialized push client is responsible for performing interactions on the replicated data. A push client has the replication logic of an ARMI client but this client has no GUI. The push client sends operations at a specific interval. The interval is based on one of the given write resolutions. The performed operations are write operations. These operations simulate a moving virtual object. These operations are the most common interactions in ARMI. With the help of the push client the write operations are sent accurate in time.

The last type of clients used for the experiment are called simple clients. A simple client has a copy of the virtual world and can receive updates from a replica manager. The simple clients are very light weight because the clients do not have the 3D interface and it cannot initiate operations on the virtual world.
6.1.5 Demographic data

Two experiments are based on human interaction on the virtual world. Ten users are invited to participate in the experiments. Ten users is enough to do the experiments and to get valuable results but it is important to realize that the user group is too small to be statistical meaningful. The influence of one single user is quite big and all the users have the same background. Doing the experiments with more users with a more diverse background was not an option because of the limited resources such as time, hardware, and available users.

The user selection for the experiment is based on a number of assumptions. Advanced computer users working on a slow system are likely to become irritated faster than an average computer user with the same system. Also, an experienced 3D gamer should find his way more quickly in the 3D interface than a computer user with basic computer knowledge. Therefore, a number of restrictions are used to select the users. The first restriction is that average computer users must be used for the experiment because ARMI is targeted at such users. The second restriction is that the users have no or little experience in software that uses 3D.

The users are asked to answer a number of questions before they start with the experiments. These answers help to get a better understanding of the population and previous experiences of the users. The following questions are answered by the ten users:

- What is your age? (open question)
- How often do you use the computer? (multiple choice)
- Are you an experienced computer user? (multiple choice)
- Are you familiar with 3D? (multiple choice)
6.1. Setup

These questions are answered by each user at the beginning of the experiments. The answers are given in Appendix B in Table B.1. From the results it becomes clear that the average age of the users is 26.2 years. The multiple choice answers are combined into three pie charts, see Figure 6.1.

![Pie charts](image)

**Figure 6.1: An impression of the computer and 3D skills of the users**

From the charts shown in Figure 6.1 it becomes clear that 70% of the users, uses the computer every day and that the rest, 30%, uses the computer a number of times per day. The third question is used to estimate whether the users are experienced computer users. Most of the users, 80%, say they have basic knowledge to work independently and the rest, 20%, say they know a few things to do their work.
The last question gives an impression about how familiar the users are with 3D. There is one user, 10%, with absolutely no experience in 3D and the rest have seen 3D (60%) or have played with 3D (30%).

It is clear that the users selection fits exactly in the given restrictions because the users have average computer knowledge and they have no or little experience in 3D.

6.2 Experiments

In this section the five experiments are described and explained. The hardware setup used for each experiment is given and what exactly is measured is described. The results of the experiments are given in “Results” (6.4).

6.2.1 The commit delay experiment

The commit delay is the time spend on processing a transaction on the majority of the replica managers. The delay is measured from building the transaction on a replica manager until it is committed on the same replica manager.

Between the creation of the transaction and when it finally commits there is time needed for collecting a majority. This time plus the time spend on creation and committing the transaction is the commit delay. This means that the commit delay is the time spend on 1) building the transaction, 2) distributing the transaction, 3) wait until a majority agrees, and 4) commit the transaction locally.

The commit delay is a good indicator of the performance of the replication logic. The commit delay is the time spent to find a consensus. For each operation performed on the virtual world a majority must be found and because of that, the commit delay is the most essential delay in the replication system.
6.2. Experiments

In this experiment the commit delay is tested with a changing replica manager configuration. Interesting to look at is the impact of different replica manager configurations on the commit delay.

This experiment starts with three replica managers and the number of replica managers is then grown to 10 replica managers. In total there are 8 different replica manager configurations used for this experiment. Per replica manager configuration five tests are performed and they differ in the used write resolution. In each test hundred operations are performed by a push client. The push client is used to perform operations on the virtual world at a given interval. The commit delay per test is the average of the hundred operations.

To run the replica managers and the push client 11 the HP Compaqs are used. There are no users involved in this experiment and the virtual world only contains the 3D hand for the push client.

6.2.2 The network delay experiment

The network delay is the total time spend on initiating an operation on the ARMI client, sending it to the replica manager, and updating the initiating client. It is important to realize that the network delay also contains the commit delay.

The network delay is the complete round trip time of an operation performed on the virtual world. The network delay contains the time spent on 1) initiating an operation on the ARMI client, 2) send it to one of the replica managers, 3) building the transaction, 4) distribute the transaction, 5) wait until a majority agrees, 6) commit the transaction locally, 7) send the transaction to the initiating client, and 8) process the transaction on the local cache of the client.

The network delay is important to measure because the responsiveness of the system depends on it. To keep the performance good and to keep the system responsive the network delay must be kept as low as possible.
6.2. Experiments

The experiment is similar as the experiment used to measure the commit delay. The replica manager configurations starts with 3 replica managers and ends with 10 replica managers. Furthermore, per configuration 5 tests are performed with different write resolutions. The hundred operations per test are performed by a push client and the network delay per test is the average network delay for the performed operations.

The same hardware setup as for the commit delay is used. In total 11 HP Compaqs are used to run the 10 replica managers and the push client. In this experiment there are no users involved and only the 3D hand of the push client is presented in the virtual world.

6.2.3 The update delay experiment

Each replica manager can handle clients and after committing a transaction on the virtual world, the connected clients must be informed about the changes. The update delay is the time used for updating the clients. Each client sends an ACK message to the replica manager when the client receives a message from the replica manager. This ACK message is used by the replica manager to calculate the RTT.

The update delay is the time spent on 1) distributing the operation to the connected clients, and 2) wait until the last client sends an ACK message. In this case the delay is not the total time spend on updating the clients but on the slowest connected client. The operations are sent concurrently and because of that, the delay is based on the slowest client.

The update delay is important because it gives information about how scalable the system is. In an ideal situation a replica manager can handle hundreds of clients without performance loss. There is no relation between the update delay and the commit delay. That there is no relation is important because this means that the number of connected clients do not have any impact on the commit speed.
6.2. Experiments

The update delay experiment is based on a replica manager configuration with 3 replica managers, one push client, and a total of 80 simple clients. The push client is responsible for performing operations on the virtual world at a given interval. This push client is connected on a different replica manager than the simple clients. The reason behind this is that the update delay measurement is more accurate because the push client uses the TCP connection also for sending the operations. This makes it a little bit heavier in terms of bandwidth than the other clients. The number of connected clients grows in steps of 10 clients. For the first step only one client is used, for the second step 10 users are used, for the third step 20 users are used, and etc. Per step five tests are performed and they differ in the write resolution used.

For this experiment the 11 HP Compaqs were used. The three replica managers and the push client run on dedicated computers. The simple clients were equally distributed on the remaining seven computers. It is reasonable to run multiple simple clients on a single machine because the influence on the measured delay is minimal because the clients have their own connection and the bandwidth used is very small. A normal desktop computer can easily run many simple clients because the clients are lightweight.

The average, maximum, and minimum update delay is measured per number of clients against the five write resolutions. Furthermore, the commit delay is measured because this is a good indication whether the number of connected clients influence the commit speed. It is important to realize that the setup itself also has a standard delay of ≈8ms. Therefore, the growing extra network delay is added to the standard network delay.

6.2.4 Responsiveness of the system experiment

The responsiveness of the system depends on how fast the system responds to user input. A user initiates an operation on the virtual world and expects that the operation is immediately performed on the data and that the changes are directly visible in the GUI. Because there are concurrent users, an operation must first be sent through the network to find a consensus.
6.2. Experiments

This replication logic costs time and this delay can frustrate users because they can believe that the system is not working properly or that an operation is not captured by the system.

When many users are involved that edit the virtual world, the network delay will eventually grow. At a given point the network delay is too high and the system is not called responsive anymore. When the scale of the system increases, the network delay must develop with respect to the responsiveness. At a given point the network delay is too high and the system is not responsive enough.

The users perform a continuous task in the 3D interface. In the 3D interface a virtual car is visible and the users are asked to place this car on the other side of the interface, see Figure 6.2. When the car is on the left side the user must move it to the right and the other way around. The reason behind this task is that the users must constantly repeat the same task which makes it possible to compare the responsiveness of the system when the network delay changes. Moving an object from one side of the screen to the other side contains all the basic functionality such as selecting and moving an object. Therefore, the task is representative for normal usage of the system.

Figure 6.2: A continuous task for the responsiveness experiment
6.2. Experiments

Per step the network delay is increased with 50ms and the first step has no extra delay (0ms). Per step the users must give their feedback about the responsiveness of the system. The last step has a network delay of 500ms which is the upper bound of this experiment because in preliminary tests the 500ms delay was found to be extremely unworkable.

The following multiple choice questions are asked per user per step:

What did you think of the responsiveness of the system?
1. Not workable
2. Quite slow
3. It is workable
4. Good
5. Very good

For this experiment the MacBook Pro and Dell XPS are used. On both machines the 3D interface and a modified replica manager is running. For this setup a single replica manager is used, which is basically a client-server approach, to eliminate the commit delay. In this specialized replica manager the network delay is artificially increased from 0ms (no extra network delay) to 500ms in steps of 50ms. This experiment is done by ten users. The users know that they must give their feedback about the responsiveness of the system. The virtual world only contains the virtual car needed to perform the continuous task.

6.2.5 AR user statistics

The clients uses the 3D interface for interaction on the virtual world. This experiment is used to discover how users use the virtual world. In this experiment the focus lies on the operations performed on the virtual world.

Building a replication system without knowledge about how users interact on the virtual world is quite difficult. AR user statistics are interesting because it gives valuable knowledge about how users uses the replicated data. Besides, it gives, for example, the number of operations performed on the virtual world per second. Perhaps specific concurrent user behaviour can be discovered which can help to improve ARMI and the AR-object replication.
6.2. Experiments

This experiment is based on users and each user performs two simple tasks in the 3D interface. For the first task the user is asked to create a landscape. This landscape must contain a house, a car, and a cow and the task is done when all the objects are visible in the virtual world, see Figure 6.3(a). For this task, the users are working in their own environment and no other users are involved. The second task is to perform a concurrent task. In the virtual world nine cows are placed at random and the users are asked to re-arrange these cows. This concurrent task is illustrated in Figure 6.3(b). The users work in pairs in the same 3D environment. The operations performed by a user in the environment are visible for both users. Each user has the same task which means that the users can work together to re-arrange the virtual objects.

![Figure 6.3(a): Arranging a landscape for single users](image1)
![Figure 6.3(b): Re-arranging cows for concurrent users](image2)

Figure 6.3: Two different tasks are used to discover the AR user statistics

Both a concurrent task and a task for single users are used to collect the AR user statistics. The reason behind this is that users may work differently in an environment with a single user or with two users. The idea behind ARMI is to support multiple users but this experiment is limited to only two concurrent users.

For the first task the MacBook Pro and the Dell XPS are used. On both machines the 3D interface and a single replica manager is running. For this setup a single replica manager is used, which is basically a client-server
6.3. Measurements

Approach, to ensure that all the interactions can be performed on the virtual world without any delay. For the second task the MacBook Pro and the Dell XPS are also used. Both machines run the 3D interface and one replica manager runs on the MacBook Pro. Both clients are connected through a 100 Mbit direct link. Working with one replica manager is important because then the user can work without any restrictions in terms of delay and real AR behaviour becomes visible.

6.3 Measurements

In “Experiments” (6.2) the five experiments are described and the used terminology is explained. In the following sections the measurements are presented.

In the commit delay experiment the average, maximum and minimum commit delay per replica manager configuration are measured for each write resolution. In total there are 8 different replica manager configurations involved. The configuration is tested under five different write resolutions. Each test is based on 100 operations to get a valuable average commit delay.

Furthermore, for the network delay the average, maximum and minimum network delay per replica manager configuration are measured for each write resolution. Which means that in total there are 8 different replica manager configurations used for this experiment. Each configuration is tested under 5 different write resolutions and for each write resolution there are 100 operations performed.

For the update delay experiment the average, minimum and maximum update delay are measured together with the commit delay. The number of clients grows in steps of 10 clients. Each step the update delay is calculated for each write resolution. In total there are five write resolutions used for this experiment. The commit delay is measured because this is a good indication if the number of connected clients influence the commit speed.

The measuring of the responsiveness of the system is not straightforward because it is based on the opinion of the users. To find out if the system is responsive, the users are asked to give their feedback on a growing
network delay. This feedback gives a nice impression about whether the system is not responsively enough for the users to work with. The feedback of the users is asked per step and in total there are 110 answers collected.

The Logging system, see “Logging system” (6.1.2), is used to log each operation performed on the virtual world to measure the user statistics. Furthermore, the initiating user is recorded which means that every operation can be linked to a user. Based on this log file the number of operations per second can be calculated and it can also be used to make normal AR behaviour understandable.

6.4 Results

In total there are five experiments, described in “Experiments” (6.2), used to evaluate the AR-object replication. In this section the most important results per experiment are briefly described. In “Discussion” (6.5) the results of the experiments are combined and used for the discussion about the AR-object replication.

6.4.1 The commit delay results

Before an operation can be performed on the virtual world the replica managers need to find a consensus. The time it costs to create the transaction on the replica manager until it is committed is measured as the commit delay.

The commit delay gives detailed information about how fast the replication logic works. The commit delay is a good indicator of the performance of the system. The commit delay is measured under a changing number of replica managers and the results are shown in Figure 6.4.
6.4. Results

In Figure 6.4 the average commit delay per replica manager configuration is shown under different write resolutions. The following write resolutions are visible: 40ms, 45ms, 50ms, 55ms, and 60ms. These write resolutions are used to perform operations on a given interval. More information about the write resolutions is given in “Used write resolutions” (6.1.3). On the y-axis the commit delay in milliseconds is given and on the x-axis the number of involved replica managers is shown.

The trend is that the commit delay is constant up to 8 replica managers for all write resolutions. When there are more than 8 replica managers involved by the replication process the commit delay grows exponentially. When the commit delay grows, the small differences in write resolutions become more visible.

In Figure 6.4 the average commit delay is shown. It is also useful to look at the maximum and minimum commit delay for each replica manager configuration. Especially the maximum commit delay is important because these delays have a great impact on the performance of the system and can be noticed by the users.
6.4. Results

In Figure 6.5 two graphs are shown. The first graph represents the maximum commit delay per replica manager configuration and the second graph gives the minimum commit delay per replica manager configuration. For both graphs the y-axis is used to give the commit delay in milliseconds and the x-axis is used for the number of replica managers. The same write resolutions are used for the average commit delay. In the minimum commit delay graph there are two moments where the minimum commit delay is 0ms which is a rounding error.

The minimum measured commit delay is \( \approx 2 \text{ms} \) and increases to \( \approx 3 \text{ms} \) for 8 and more replica managers. The maximum commit delay per replica manager configuration shows the same trend. That problems arise from 8 replica managers is clear. In this graph the maximum delays increase, for most of the write resolutions, to 400ms. Which means that there is at least one transaction per write resolution that is committed with a delay of \( \approx 400 \text{ms} \).

6.4.2 The network delay results

The network delay is the time it costs to perform an operation on the virtual world from the user’s perspective. It is a complete round trip from initiating an action on ARMI, until the operation is performed on the local copy of the client. The network delay contains the time spend on communicating with the replica manager as well as the commit delay of the setup.
6.4. Results

Figure 6.6: The average network delay per replica manager configuration

The average network delay is the most important delay in the 3D interface and it illustrates how much time each interaction costs on the virtual world. The network delay itself is crucial to illustrate how the users perceive the responsiveness of the system. The average network delay is shown in Figure 6.6.

The graph in Figure 6.6 illustrates the average network delay for different replica manager configurations per write resolution, see “Used write resolutions” (6.1.3). The following write resolutions are used: 40ms, 45ms, 50ms, 55ms, and 60ms. The number of replica managers involved in the replication process is visible on the x-axis and the network delay in milliseconds is given on the y-axis.

In Figure 6.6 it is visible that the network delay is constant up till 8 replica managers and all the write resolutions perform the same except for the 40ms write resolution. The network delay grows rapidly when a replica manager configuration is used with more than 8 replica managers.

The average network delay gives an impression about the time it costs to perform an operation by a client. However, an average value is not always a good representation of the reality because big fluctuations in the network delay can be annoying for the users. Therefore, the maximum and minimum network delays are shown in Figure 6.7.
Both maximum and minimum network delays are shown in Figure 6.7 for each replica manager configuration. The first graph represents the maximum network delay and in the second graph the minimum network delay is given. The y-axis is used to give the network delay in milliseconds and the x-axis is used for the number of replica managers for both graphs in Figure 6.7. The same write resolutions are used as displayed in the previous results.

The different write resolutions globally performs the same until a replica manager configuration is used with more than 8 replica managers. The same trend is visible as in the average network delay.

### 6.4.3 The update delay results

The update delay is the time it costs to update all connected clients. Each client has a local copy of the virtual world and when the replica manager commits a transaction, this transaction must also be performed on the client’s copy of the data. The replica managers sends each transaction to all the connected clients. The time between starting to inform the clients about an update and when the last client is up to date is called the update time.

The update delay is measured for only one replica manager configuration with 3 replica managers because the delay is independent of the number of replica managers. For this experiment the write resolutions earlier discussed
are used to look if there are any relations between the write resolutions and the update delay with respect to the number of clients, see the graphs in Figure 6.8.

**Figure 6.8:** The update delay under different write resolutions
6.4. Results

In Figure 6.8 one graph is shown for each write resolution. The y-axis is used to show the update delay in milliseconds and the number of connected clients is given on the x-axis. In each graph there are four lines used. The red line illustrates the average update delay, yellow is used for the minimum update delay, green describes the maximum update delay, and the blue line is used to describe the measured commit delay.

From the graphs in Figure 6.8 it becomes clear that for each write resolution the same trend is visible. For each write resolution the average update delay starts at $\approx 1\text{ms}$ for one connected client and grows up to $\approx 7\text{ms}$ for 80 clients. Because the same trend is visible for all the write resolutions, it is clear that the write resolution has no impact on the update delay. The maximum update delay and minimum update delay fluctuate a little bit but not very extreme.

Because the different write resolutions have no impact on the update delays, it is safe to combine the results of the different write resolutions and calculate the average update delay. This average update delay gives more accurate information about how the update delay developed based on the number of clients, see Figure 6.9.

![Figure 6.9: The average update delay](image-url)
6.4. Results

In Figure 6.9 the average update delay, the average maximum delay, the average minimum delay, and the average commit delay is shown. Again, the y-axis is used for the update delay in milliseconds and the x-axis is used for the number connected clients. The same four lines as in the previous graph are used to represent the different update delays and the commit delay.

From Figure 6.9 it is clear that per 10 users an extra update delay of ≈1ms is added and that the development of the update delay is linear. Another interesting fact is that the commit delay, shown in blue, is not affected by an increasing number of clients or a growing update delay. Which means that the number of clients does not influence the commit delay.

6.4.4 Results responsiveness of the system

A system is called responsive when an operation, initiated by a user, is performed as fast as possible without an annoying delay for the user. The responsiveness depends on how the user experiences the network delay. When the network delay is low the system can quickly handle the operations and no delay is noticed by the users. When the network delay grows, more time is used between initiating the operations and when the performed operation is visible to the user. In that case the system is slow to respond on interaction and the system is not responsive.

Responsiveness per network delay

Measuring the responsiveness of the system is not easy because it depends on how the users think about the process speed of the interactions. The users for this experiment are asked to give the feedback on a artificially growing network delay.

The responsiveness is important for the AR-object replication because it is a (hard) limit in the scalability of the system. The system is scalable until the system is not responsive anymore according to the users. Because of that the responsiveness is important and especially how the responsiveness develops according to the network delay.
6.4. Results

Figure 6.10: The feedback about the responsiveness per network delay

The responsiveness per network delay is shown in Figure 6.10. On the y-axis the percentage of the users is visible. The range starts with 0%, which means 0 users, and ends with 100% which means ten users. The network delay is visible on the x-axis and starts at 0ms, which means no extra network delay, and stops at 500ms. The network delay is increased with 50ms each step. The colors represent the feedback of the users. Green is used as Very good, light-green is Good, Yellow is Workable, Orange is used for Quite slow, and finally red is used to represent a Not workable situation.

It is interesting to see that 20% of the users describe a delay of 0ms as Good and the rest as Very good. This illustrates the problem in asking feedback from the users. Each user has his own definition of Good and Very good. Because no delay is used you expect that all the users say that the responsiveness is Very good. Another interesting fact is that some users are really fast in calling the system Quite slow. In one case the user jumps from Good to Quite slow. From Figure 6.10 it also becomes clear that a few users are very optimistic about the responsiveness of the system and they call a system with a network delay of 300ms workable, which is difficult to believe.

The responsiveness score

The responsiveness score is used to give the average responsiveness per network delay. This average responsiveness is called a score because it is based on all feedback of the users. With the help of the responsiveness score it is possible to relate, for example, the different responsiveness definitions
to a specific network delay. The responsiveness score is important because it can help to define the maximum network delay to deliver a system with a given responsiveness requirement.

Figure 6.11: The responsiveness score per network delay

The average responsiveness score is shown in Figure 6.11 with a purple colored line. Also the responsiveness definitions are visible in the same colors as in Figure 6.10. On the x-axis the extra network delay is given in milliseconds and on the y-axis the average responsiveness score is shown. The mean score is 0 when the system is called workable and a positive number indicates a better system in terms of responsiveness and a negative number indicates a slower system according to the mean score.

The score is calculated based on the following constants: Not workable is -10, Quite slow is -5, It is workable is 0, Good is 5, and Very good is 10. For each network delay the answers of the users are multiplied with the constants. Which means that if every user thinks that a network delay of 0ms is Very good the score is 100 because 10 users x 10 is a score of 100. This score for each responsiveness definition is visible in Figure 6.11.

In the average responsiveness score graph, Figure 6.11, it is shown that the responsiveness score decreases fast. The responsiveness of the system becomes worse rapidly according to the users. Most of the users think that 100 ms network delay is Workable.
The trend is that in the beginning, with a low network delay, most of the users give the same feedback but later on, when more and more network delay is added, the feedback from the users becomes more diverse.

6.4.5 AR user statistics

In this section the results are given for the AR user statistics experiment. AR user statistics are used to get a better understanding about how the users are using the virtual world. Especially the read/write ratio is interesting and how the number of operations develop according to the time.

Read/update resolutions single users

Recall from “AR user statistics” (6.2.5) that the AR user statistics experiment is used to discover normal AR user behaviour. This experiment contains two different tasks and the first task is for single users and the second task is designed for concurrent access to the virtual world. First the results are given for the task with only single users.

Ten users have done the experiment. The first task was to design a landscape with three virtual objects. This part of the experiment is interesting because it gives information about performed operations on the virtual world for a single user.

In Figure 6.12 the number of read and update operations per user for the first task are given. The read operations are used by ARMI to read object data from the virtual world and the update operations are used to alter the virtual world. This includes create, write and delete operations. The reason for combining these operations in the results is that most of the operations are write operations and only a few operations are create and delete operations.
6.4. Results

In total there are nine graphs given in Figure 6.12, for each user one. On the y-axis of each graph the number of operations is shown and on the x-axis the time is given in seconds. There are periods, for example for User 1 between 160 and 200 seconds, that no update operations are performed. Each flat blue line represents such a period. This flat blue line means that the 3D interface is not used by the user and no interactions are performed on the virtual world.

![Graphs for User 1 to User 10](image)

Figure 6.12: The read/update operations per user for the first task

Based on the results the average read/write ratio is $\approx 10:1$ for single users. Which means that for each update operation there are 10 read operations performed by ARMI.
6.4. Results

Read/update resolutions concurrent users

The second task of the AR user statistics experiment is based on concurrent users. The users work pairwise on the virtual world to re-arrange a landscape. The landscape contains a number of virtual objects and the users are asked to create groups of virtual objects. The initial idea of ARMI is that multiple users can work together and perform simultaneous operations on the virtual world. The AR user statistics for multiple users is interesting because it gives detailed information about the read and update resolutions used by concurrent users.

Figure 6.13: The read/update operations per user pair for the second task

In Figure 6.13 the read and update operations per user pair for the second task are given. Also for this graph the write, create and delete operations are combined into one value, called the update operations. In total there are 5 graphs visible in Figure 6.13, for each user pair one.
6.4. Results

The y-axis is used for the number of operations and on the x-axis the time in seconds is given. In each graph three lines are visible. The first line, given in blue, represents the update operations processed on the virtual world by both users. The second line, in red, shows the number of reads per user and the third yellow line is used to illustrate the total reads performed on the virtual world for both users.

For this task the users work together in pairs. The virtual world already contains a number of virtual objects. Because the task is about re-arranging virtual objects no new virtual objects are created during this part of the experiment. This influences the trend of the read operations per users. The number of reads is quite constant which again illustrates the relation between the number of objects in the system and the performed read operations. Each user has a 3D interface, which means that the total read operations for the system also grows according to the number of users.

The read/write ratio for this concurrent part of the experiment is $\approx 25:1$ which is much higher than the read/write ratio of single users.

In Figure 6.14 the number of update operations in seconds is visible for the user pairs. The y-axis of the graph shown in Figure 6.14 is used for the number of update operations and the x-axis is the time in seconds. Some user pairs are a lot faster than other. For example user pair 4 is finished in 100 seconds and user pair 1 uses more than 250 seconds to finish the task.
6.4. Results

Figure 6.14: The update resolution of user pair 1, 2, 3, 4, and 5

It seems that there is no relation between the number of update operations per second and the time used to finish the task, see Figure 6.14. User pair 4 is very fast and uses fewer update operations for re-arranging the virtual objects than the pairs that work longer on the task. The slower user pairs use more update operations to perform the same task. This means that the update operations, in case of the slower user pairs, are not used very efficiently. In Figure 6.14 it is visible that the number of update operations per second constantly changes. On several occasions the number of operations per second grows fast and also 3 spikes are noticeable with more than 50 operations per second, see Figure 6.14.
6.4. Results

Write access to virtual objects

The AR user statistics experiment described in “AR user statistics” (6.2.5) contains two tasks and one of the tasks explores the concurrent behaviour of AR users. This data is used to investigate how AR users interact with the virtual world. The logging system is used to record all the operations performed on the virtual objects. The graph shown in Figure 6.15 illustrates how often a virtual object is edited by more than one user.

This data is interesting because it gives an impression about how often a conflict arises on object level. When multiple users edit the same virtual object at the same time it is likely that more conflicts arise than when only one user changes the virtual objects.

![Figure 6.15: The number of operations by different users on the same object](image)

The graph in Figure 6.15 is used to represent how often a virtual object is edited by more than one user. The y-axis shows the number of operations in percentage and the total number of operations performed on an object is 100%. In total 55 virtual objects are used for the experiments, for each user pair are 9 virtual cows and two virtual hands used. The virtual objects are numbered on the x-axis. User 1 in blue and User 2 in orange represents distinct users and are not related to specific users. Objects which are only blue or orange are objects that are edited by one user. Objects with both colors represent objects that are edited by multiple users.
In total 55 objects are used and only 5 objects are edited by multiple users. For the objects, that are edited by more than one user, there is also one main editor. This main editor is responsible for most of the operations performed on the given object.

6.5 Discussion

The results of the experiments are given in the previous section “Results” (6.4). In this section the results of the experiments are discussed. The graphs, used for representing the results, show curious behaviour in specific situations and for this behaviour an explanation is given in the following paragraphs.

The graph shown in Figure 6.4 clearly shows that the commit delay for replica manager configurations with 8 or fewer replica managers is \(\approx 5\text{ms}\). Which means that the write resolution of 40ms, the most heavy write resolution used in this experiment, can be handled. The problems arise from 9 replica managers onwards. For each write resolution the commit delay grows fast from a delay of \(\approx 5\text{ms}\) to \(\approx 100\text{ ms}\). It looks like the slowest write resolution of 60ms, has less impact but the commit delay still grows exponentially. From Figure 6.4 it is clear that all the different write resolutions have problems with 9 or more replica managers and that the used write resolution is not directly responsible for the exploding commit delays.

The most likely explanation is that the participating replica managers give their vote power at the same time. One replica manager initiates a transaction and sends it through UDP multi-cast to the other replica managers. The other replica managers are constantly waiting and when a transaction request arrives, a response is given immediately by all replica managers through UDP at the same time. It is likely that other replica managers give their response at the same moment because the replica managers run the same software, have the same hardware, and wait in the same state.
6.5. Discussion

This excessive UDP broadcast introduces collision problems. This is the most likely explanation for the exploding commit delay. Only a majority of the voting power is needed to commit a transaction but under these circumstances the majority is not reached at all copies because messages are not arriving. To repair these situations a lot of syncing is needed and this heavily influences the commit speed.

In Figure 6.6 it is visible that the network delay is constant up till 8 replica managers. From the commit delay results it is clear that exactly at the same number of replica managers the commit delay is growing. Because the commit delay is part of the network delay, the same trend is visible. All the write resolutions perform the same, except for the 40ms write resolution. According to the other write resolutions the 40ms variant is slow with a network delay of \( \approx 23\text{ms} \). It looks like there is not always enough time to distribute the operations to the clients. The next operations already arrive but the replica manager is still working on the distribution of the previous operation(s).

From Figure 6.7 it becomes clear that extreme network delays are measured. Interesting to see is the spike around the replica configuration of 9 replica managers for both 40ms write resolution and 45ms write resolution. This effect is visible in the average network delay, the minimum network delay, and the maximum network delay. Because this spike does not fit in the trend, which is growing, and the fact that this pike is visible in all three graphs, it seems that this pike is the result of an increase of other network traffic, for a short period of time, or something else on the network that influences the network speed.

The update delay is an important aspect in the scalability of the system. The less time is needed to update the connected clients, the more clients a replica manager can handle. From Figure 6.9 can be concluded that the time delay grows linearly with the number of connected clients. Each increase of 10 users will introduce an extra update delay of \( \approx 1\text{ms} \). The linear development of the update delay was expected but adding more clients is cheaper than expected. This can be explained by how the replica manager handles the connected users. Often threads are used to handle the connections to the users because it is simple. Another technique,
used in this prototype, is based on the select() system call in the I/O library. With the help of this technique it is possible to handle multiple communication channels at once. The I/O is done on the background so that the program can do other things in the meanwhile. With this technique each user costs $\approx 0.1\text{ms}$ extra update delay. In Figure 6.9 the minimum and maximum update delays are also shown. These delays have an oscillatory behaviour. In most of the cases the maximum network delay develops the same as the minimum network delay. Therefore, it looks like a temporary increase of other network traffic is responsible for this behaviour.

The responsiveness of the system depends on the network delay. The users are asked to give their feedback on different network delays and the results are shown in Figure 6.10 and Figure 6.11. Based on this results it is clear that the problems arise after $\approx 100\text{ms}$ network delay. When the network delay is bigger than 100ms the system is not responsive according to the users. Therefore, to keep the system responsive the network delay must be smaller than $\approx 100\text{ms}$. The users have no experience with ARMI and have only worked for a few minutes with the system. It is questionable if the feedback given by the users is accurate. Not only the users have their own definitions about Very good, Good, and etc. but also about the fact that the environment is still very new for the users. What if the users work with this product every day? It is reasonable that in such case the system is called Not workable a lot faster than shown in Figure 6.10. The best network delay, with respect to the responsiveness, lies around 50ms or lower because all the users say the responsiveness of the system is Good or Very good. Therefore, to get a proper responsive system the operations must be performed under 50ms.

Each user has performed the same task for the experiment. This task is used to investigate what the AR interactions looks like. The number of update operations and the read operations are shown in Figure 6.12. There are two interesting behaviours visible. The first one is that some users finish a lot earlier than the other users, which is visible in the x-axis. These users work longer on the task but also perform a lot more operations. More logical would be that, on average, the same number of operations are needed to finish the task. The reason for this behaviour is that the operations performed on virtual objects depend on the input device. There
is a relation between the virtual object and the input device. Therefore, each movement of the input device is translated in an update operation for the virtual object. Because a mouse is used as an input device even the smallest movement of the mouse is used as a trigger to edit the virtual object. When the mouse moves slow on the screen it means more updates are performed than when the mouse moves fast across the screen.

The second interesting behaviour, visible in Figure 6.12, is the growing number of read operations. The reason for this is that during the task more and more objects are created and this influences, of course, the number of reads. Which means that the number of read operations depends on the number of virtual objects. For this task with single users the read/write ratio is \( \approx 10/1 \) but it heavily depends on the number of objects in the virtual world. The same trend is visible in the graph used to represent the results for the second task which is based on concurrent users, see Figure 6.13. Because this task is about re-arranging virtual objects that already exist in the virtual world, the number of read operations start very high and the measured read/write ratio is \( \approx 25:1 \). From both tasks it is clear that the read/write ratio depends on the number of objects presented in the virtual world. This behaviour is responsible for a high read/write ratio in the case of a virtual world with many virtual objects.

At several graphs shown in Figure 6.13 small drops in performed read operations are shown. In such cases the number of virtual objects is decreased. A disconnected user is responsible for this behaviour. In these situations the users lose their virtual hand in ARMI because sometimes the users move their hand outside the screen. In that case the client is reconnected to the virtual world. Which means that a virtual hand is first removed and than a new virtual hand is created for the new user. This influenced the read operations because the number of virtual objects has changed.

From this AR statistics experiment it is clear that the number of read operations is based on the number of virtual objects and the number of connected clients. The discussed read/write ratios are based on a large set of data and thus representative. However, the problem with an average read/write ratio is that extreme values are not visible. The update operations performed are changing each second, see Figure 6.14. Three spikes are visible about
50 operations per second and also a number of short bursts of operations are shown. The reason behind these bursts and spikes is that the users are trying to finish a sub task very quickly. For example, grabbing an object and move it directly to a new location.

It is quite surprising that most of the objects (91%) are only edited by a single user, see the graph in Figure 6.15. The rest of the objects (9%) are edited by more than one user. For the five objects, that are edited by multiple users, there is always one user that performs most of the operations on a given object. The virtual objects are represented as objects in the reality through AR but users work on these objects just as real physical objects. For physical objects, for example a tea cup, it is not likely that more than one human interact with it at the same time. Users are handling virtual objects exactly in the same way. Which means that the interaction rules for physical objects are also used for the virtual objects. This explains why users are not interested to interact on a virtual object which is already ’owned’ by another user.
Chapter 7

Conclusion and Future work

In this thesis an AR-object replication approach is presented which is part of ARMI. ARMI is a prototype built on commercial off-the-shelf (COTS) hardware to explore the possibilities of AR. Through a Head Mounted Display (HMD), with a built-in camera, the concurrent users can see the virtual maquette in reality and users are able to interact on virtual objects presence in the maquette.

The AR-object replication is located between the 3D interface and the virtual world. The 3D interface is used by the users to interact with the virtual world through the AR-object replication layer. Because multiple users are involved, object replication is needed to give each user read and write access to the virtual world. The biggest challenges are the number of read operations performed by the 3D interface and that the 3D interface cannot wait until the data is verified because of performance reasons.

To solve the replication problem an asynchronous quorum algorithm is chosen with a simple but effective cache. The quorum used for the AR-object replication is based on majority consensus, which means that for every operation performed on the virtual world a consensus must be found. To get a consensus at least a majority of the participating copies must agree on the operation. Quorum algorithms can improve the availability of the system and can handle copy and communication failures. Strong consistency is used for the virtual world to ensure that interactions, initiated by the concurrent users, looks as realistic as possible.
In total two types of consistencies are used in the system. Every 3D interface has a cache which is a local copy of the virtual world. This copy is used for fast reading and can be stale. The second type copy is located at the replica managers. The strong consistency is guaranteed for each operation performed on the data. An optimistic speculative variant is used to improve the replication speed. This variant is called speculative because even when operations are conflicting they still collect votes and thus make progress. Synchronization is based on a technique called progressive synchronization. With the help of piggybacking other copies of the virtual world can learn that a specific copy is outdated. Even before a copy realizes that he is outdated, other copies can already send sync messages to fix the copy.

Experiments are used to test if a solution towards the problem statement is built. In the following sections the main results are given. The results are used to answer the question whether the chosen approach is right. The users for the experiment give their feedback on a growing network delay which is the most crucial delay in the system in terms of responsiveness. To keep the system responsive the network delay must be smaller than about 50ms. Which means that the system is responsive when the operations on the virtual world are processed within 50ms. This limit is the first interesting result from the experiment.

The development of the network delay is quite surprising. To measure the network delay different replica manager configurations are tested and interactions are simulated by operations performed at a given frequency. This frequency is called the write resolution and the used write resolutions are 40ms, 45ms, 50ms, 55ms, and 60ms. Up to 8 replica managers the measured network delay is smaller than 10ms for all write resolutions except for the write resolution of 40ms. For this write resolution the average network delay is ≈23ms until more than 8 replica managers are used. Each 3D interface is connected to a replica manager and the replica manager is responsible for updating the interface. One experiment is used to discover the impact of clients on the network delay, the so called update delay. Every extra 10 clients introduces an extra delay of about 1 ms. The development of the delay progresses linearly for each replica manager. Which means that when 30 interfaces are connected and equally distributed across the three replica managers, the maximum update delay is 1ms.
As stated in the problems statement, the AR-object replication must have the following features; strong consistency, high availability, scalability, can tolerate copy and communication failures, no ownership or locking, minimize the size of the communication messages, synchronization must be fast, and finally, the system must be responsive enough for users to work with. Strong consistency and the availability are enforced by the Majority consensus algorithm. This algorithm can also handle copy and communication failures. No ownership or locking is used and progressive synchronization is used to repair outdated copies as fast as possible. Because the size of the operations are smaller than the maximum size of a UDP packet only a simple delta compression technique is used. Extra compression will introduce extra unnecessary delays. In a system setup with, for example, three replica managers and two 3D interfaces, the system is responsive because the system can handle the write operations within 50ms which is the maximum network delay to keep the system responsive.

It is questionable if this setup is really scalable. The system is scalable up to 8 replica managers. When more replica managers are added the network delay grows rapidly. Each replica manager can easily handle a dozen clients without serious impact on the network delay. Therefore, the system is scalable in terms of the distribution of operations to the clients. The system can also handle the increase of read operations per interface because of the local copy of the virtual world. However, the update operations are more problematic. A single user produces on average $\approx 10$ update operations per second and two concurrent users performs $\approx 14$ update operations per second. Based on these numbers the system should be able to handle the update operations because the measured network delay is smaller than 10ms.

With this network delay a theoretical throughput in terms of update operations is $\approx 100$ per second. The problem is that this number is not realistic because of two reasons. The first reason is that the network delay depends on the speed of the network and this fluctuates heavily. The second reason is that the users are very unpredictable in terms of operations performed each second. During the experiments with only two users, spikes are measured of almost 50 operations per second. Which means that, when more than two users are involved, even more extreme update bursts are likely to happen.
7.1. Future work

The presence of update bursts in combination with a fluctuating network delay makes it very difficult to keep the system responsive enough for many concurrent users.

The strong consistency, scalability, and availability required for the virtual world is difficult to achieve because of the nature of the 3D interface and how the users interact with the virtual world. In situations where strong consistency is important, the relaxation of the scalability and availability parameters is needed to keep the system responsive. When scalability and availability are more important it is better to choose for a more optimistic replication approach and thus relaxing the strong consistency parameter.

Designing and building a distributed system without a good debug strategy is the number one pitfall. For building a reliable distributed system good debugging is crucial. Realize that global state information is important. Not only to localize bugs but also to get a better understanding of the system as a whole. Last but not least, it is very useful in improving a system. Therefore, a good debugging strategy should be an important aspect in designing a distributed system.

7.1 Future work

In this thesis a proof of concept AR application is designed and built. This resulted in a number of improvements and future work.

Most of the operations on an object are performed by one user (> 90%). Which means that most of the time the data of an object is changed by one user. This behaviour can be used to improve the object replication speed by changing the consistency level. In the current prototype the complete virtual world is consistent and all the operations on all objects are serialized with respect to each other. Based on the results, most of the files only have one editor. Changing the consistency to object level looks promising in order to improve replication speed.
The problem statement is responsible for a pessimistic replication approach because one of the replication parameters is strong consistency. In the proof of concept the virtual world is strong consistent but for the clients a local copy is used for reading. This last copy is not guaranteed to be up to date for each read operation. Based on the results of the AR user statistics experiment it is now clear that conflicts on object level rarely happen. A more optimistic approach will certainly improve the situation. The downside is, of course, that if there are conflicts they are fixed afterwards which makes it less realistic.

In general it is safe to say that AR applications are mobile applications and thus weakly connected. The experiments are performed using the network of the University of Groningen. This means that the results cannot be used to give an impression about the speed when mobile devices are used. More research is needed to keep the virtual world consistent when the participating nodes are weakly connected.

The AR-object replication is vulnerable for update bursts and more research is needed to control these bursts. For example with a maximum number of update operations per user per second or to look at alternative update operations based on direction and speed instead of constant updating a new location of the virtual objects. The game industry has already answered these questions because the first person shooter game Quake is playable even when there is a network delay of > 150ms [Arm03b].
7.1. Future work
Bibliography


Appendix A

Network delay cloud computing

Cloud computing looks promising for making AR-object data scalable and available. An experiment is used to get an impression about how much time it costs to interact with virtual objects located inside Google’s cloud. The two main operations are reads and writes. Each operation is performed ten times on the cloud and the average delay of the operations is calculated.

For this experiment a client and a server are built. The client runs locally and interacts with the virtual objects through the server which runs on Google’s cloud. The virtual objects are stored in Google’s Bigtable. The experiments were done from three different locations and three types of connections were used. The first connection method is reconnecting every time an operation is performed and the second method uses a persistent HTTP connection. The last connection type also uses a persistent connection as well as Gzip encoding. The results are shown in Figure A.1.
Figure A.1: The results of the performed operations on the cloud

From Figure A.1 it is clear that performing operations on Google’s cloud is time-wise quite expensive. From a dedicated server in the USA the read operations are performed in $\approx 45$ms and writing costs $\approx 110$ms. This experiment is statistically not meaningful because only 10 operations are performed per session. To get a complete overview of the performance many more experiments are needed. On the other hand, the results give an idea about the performances of Google’s cloud. It is clear that the scalability and availability comes with a price and the delays are disappointing.

The AR-object replication must be responsive which can be quite difficult with such a network delay. Google’s cloud is interesting and easy to use but the cloud is not useful for AR-object replication because the system itself is too slow. Furthermore, the RTT from users to the cloud is too high to get a responsive system.
Appendix B

Answers on openings questions

Before the experiments the users were asked to give answers on a number of questions to get an idea about the user selection. In Table B.1, shown on the next page, the answers are given on the open and multiple choice questions.
<table>
<thead>
<tr>
<th>Questions</th>
<th>Answers</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. What is your age?</td>
<td>avg: 26.2</td>
</tr>
<tr>
<td>2. How often do you use the computer?</td>
<td></td>
</tr>
<tr>
<td>a) A number of times per day</td>
<td>30%</td>
</tr>
<tr>
<td>b) Once a day</td>
<td>70%</td>
</tr>
<tr>
<td>c) 1-3 Times a week</td>
<td>0%</td>
</tr>
<tr>
<td>d) 1-3 Times a month</td>
<td>0%</td>
</tr>
<tr>
<td>e) 1-3 Times a year</td>
<td>0%</td>
</tr>
<tr>
<td>3. Are you an experienced computer user?</td>
<td></td>
</tr>
<tr>
<td>a) No, I am limited in computer knowledge</td>
<td>0%</td>
</tr>
<tr>
<td>b) No, I know a few things to do my work</td>
<td>20%</td>
</tr>
<tr>
<td>c) I know the basics to work independent</td>
<td>80%</td>
</tr>
<tr>
<td>d) Yes, my knowledge is above average</td>
<td>0%</td>
</tr>
<tr>
<td>e) Yes, I am an experienced computer user</td>
<td>0%</td>
</tr>
<tr>
<td>4. Are you familiar with 3D?</td>
<td></td>
</tr>
<tr>
<td>a) No, I am not familiar with 3D</td>
<td>10%</td>
</tr>
<tr>
<td>b) No, I have seen 3D but I have no 3D experience</td>
<td>60%</td>
</tr>
<tr>
<td>c) I have played with 3D but I have no experience</td>
<td>30%</td>
</tr>
<tr>
<td>d) Yes, my experience with 3D is above average</td>
<td>0%</td>
</tr>
<tr>
<td>e) Yes, I am an experienced 3D user</td>
<td>0%</td>
</tr>
</tbody>
</table>

Table B.1: The computer and 3D skills of the users