
Masterarbeit

Gerald Melles

# Towards Distributed Interaction Systems in the Virtuality Continuum

*Fakultät Technik und Informatik*
*Department Informatik*

*Faculty of Computer Science and Engineering*
*Department Computer Science*

Gerald Melles

# Towards Distributed Interaction Systems in the Virtuality Continuum

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Kai von Luck
Zweitgutachter: Prof. Dr. Philipp Jenke

Eingereicht am: 9. August 2019

**Gerald Melles**

**Thema der Arbeit**

Towards Distributed Interaction Systems in the Virtuality Continuum

**Stichworte**

Virtuelle Welten, Interaktion, verteilte Systeme

**Kurzzusammenfassung**

Diese Arbeit untersucht eine Reihe von Interaktionsobjekten in Hinsicht auf die Anforderungen, welche für ihre Integration in eine Anwendung des Virtuality Continuums (VC) erfüllt werden müssen, basierend auf Experimenten und Machbarkeitsstudien im Creative Space for Technical Innovations (CSTI) und anderen wissenschaftlichen Kontexten. Damit dient sie als Ausgangspunkt für die Entwicklung umfassender Integrationslösungen für verteilte Interaktionssysteme und als Basis für deren Vergleich.

**Gerald Melles**

**Title of Thesis**

Towards Distributed Interaction Systems in the Virtuality Continuum

**Keywords**

Virtuality Continuum, VR, AR, interaction, distributed systems, ultimate display

**Abstract**

This thesis explores a range of interaction objects in terms of the requirements their integration into a virtuality continuum (VC) application would pose, based on experiments and feasibility studies in the Creative Space for Technical Innovations and elsewhere. It thereby acts as a starting point for the development of comprehensive solutions for the integration of distributed interaction systems, as well as a basis for their evaluation.

# Contents

# List of Figures

# 1 Introduction

Virtuality systems - that is, systems conceptually located anywhere within the virtuality continuum (VC, fig. 1.1, see [51]) - have seen a steady increase in interest of the research, industrial and public sectors over the last few years. New use cases are proposed nearly every other day. Virtuality continuum applications are increasingly seen not only as a toy but also as a tool, to be used in computer gaming as well as in diverse fields such as teaching, surgery, elevator repair or elder care. That said the overall adoption of virtuality systems remains fairly low.

In order to gain further insights into this new generation of human-computer interaction systems, researchers commonly need to develop their own VC applications. Of the many challenges faced by researchers during these development projects, many seem to originate from the lack of established methodologies and tools for their use cases. This is the case particularly if the application being developed differs significantly from the state of the art. The tools and methodologies provided by the industry are typically intended for either very specific use cases, or those use cases which are simply most commonly encountered. Therefore, existing tools are often insufficient for simulations developed in research contexts.

If the application's requirements include a need to integrate devices and services outside of the local hardware attached to and software running on the host system, developers are often required to create their own integration solutions. Yet these distributed devices and services may offer considerable potential as objects and facilitators for user interaction, making it an area of interest to researchers. Even where an implementation has been created by researchers, its re-use is often difficult, especially in the differing technological ecosystems of other research contexts.

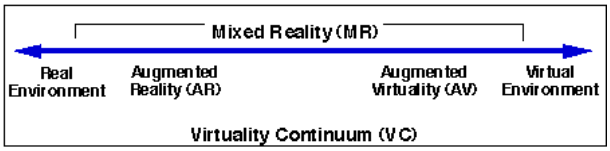
Contemporary research into virtuality technologies concentrates on individual techniques and technologies and their effect on human participants. The vast majority focuses on a single enabling technology or interaction design. Approaches for the efficient integration of interaction objects into the simulation applications of research contexts are rarely examined. In the proceedings of the 2018 IEEE VR conference for example, of the more

than 250 accepted entries not a single paper deals with such integration aspects. It is this lack which this thesis aims to address.
This thesis aims to extend the set of interactions offered by VC simulations, particularly in terms of the objects of interaction available to the simulation's participants. It proposes the integration of distributed systems and devices with the VC system as a means of achieving this. To this end, a number of feasibility studies have been conducted within a typical research context. This thesis shows how such integrations may be achieved and discusses and catalogues their requirements and challenges. It bases its analysis on the conducted studies as well as a systematic literature review.
The goal is to provide building blocks for the development and evaluation of distributed interaction systems - frameworks for the comprehensive integration of distributed interaction objects.
The project reports [45] and [44] form some of the foundation of this thesis, as does the Omniscope project paper [46] and the experiments detailed in the thesis' appendix.

## 1.1 Virtuality Continuum

In [51], Milgram and Kishino proposed the virtuality continuum as a taxonomic basis for artificial, virtual and immersive experiences using wearable displays (head-mounted displays, HMDs). The term has since been established as a way to distinguish between Virtual, Augmented, Blended and Mixed Reality applications.
This distinction can be somewhat indistinct. For instance, there are no clear distinctions for when a simulation could be seen as augmented virtuality rather than augmented reality. Rather, the taxonomy serves to illustrate the existence of a spectrum in general. It is also important to bear in mind that the taxonomy had only been intended for visual displays - only visual stimuli from the simulation to the user are taken into account. Modern applications which are considered part of the VC are likely to feature additional devices and stimuli which cannot necessarily easily be accounted for in this model. They also commonly feature a number of feedback channels which the taxonomy similarly does not directly mention.
Over the decades since its conception, the term virtuality continuum has been interpreted and re-interpreted repeatedly. In [9], for instance, the term reality-virtuality continuum was used instead. The same often holds true for illustrations of the continuum - in [9], their illustration diverged similarly from the original as their terminology did.
In order to avoid such ambiguities and limit the possibility of misinterpretation, the

Fig. 1.1: Milgram and Kishino's Virtuality Continuum ([51])




following paragraphs will illustrate how the term Virtuality Continuum as well as its constituent parts are used in this thesis.
The term Virtual Reality (as well as its acronym VR) is often used as an umbrella term encompassing some or all of the virtuality continuum. In this paper, the term Virtuality Continuum (or VC) will be used instead, to ensure clear distinctions where necessary. In [28], Sherman and Craig define virtual reality as follows: "a medium composed of interactive computer simulations that sense the participant's position and actions and replace or augment the feedback to one or more senses, giving the feeling of being mentally immersed or present in the simulation (a virtual world)". This serves our need for a broad definition encompassing all of what will be referred to as the virtuality continuum in this thesis. It is noteworthy, however, that our definition differs slightly from the template set by Milgram and Kishino: A fully 'real' environment will not be considered part of the virtuality continuum. Consequently, the term virtuality continuum application for example would not include an application which did not augment or replace any sensory feedback to its users.

**Virtual Reality** As mentioned above, in some works such as [28], virtual reality is defined as encompassing all other types of virtuality applications. However, in order to be able to reflect some of the parts this spectrum consists of, the term virtual reality will be used in this thesis only to refer to those simulations which predominantly replace rather than augment feedback to the participants' senses. By this definition, it would for example include most virtual reality games available for consumer-grade head-mounted displays such as the HTC Vive or Oculus Rift. It would, however, not include the use of a Microsoft HoloLens HMD for assisting in elevator repairs (see [78]).

**Augmented Reality** Augmented Reality (AR) describes those simulations which use virtual effects comparatively sparingly, to augment those stimuli originating from the real world. It has become customary to call those VC applications augmented reality or AR applications which add virtual (computer-generated) visual stimuli to real ones (from the

user's environment). Microsoft's HoloLens serves as a good example for this: The user can see their real surroundings through transparent glasses, but the HoloLens is capable of creating 'holographic' illusions through the glasses which make virtual entities appear as though they had a physical form inside the real world rather than in an entirely virtual environment.
It should be noted that the most common and prevalent type of augmented reality application is that of smartphone based augmented reality. These applications work by using the smartphone's screen (front) and camera (back) together to create the illusion of a small 'window' - as though the cellphone was nothing but a frame. On this, simulated content can then be overlaid. There are many applications for smartphones which utilize this effect, ranging from entertaining to genuinely useful.: Pokemon GO, for example, lets its users hunt, train and trade virtual creatures while the Pattarina application leverages a smartphone-based AR toolkit to overlay sewing patterns onto fabric, negating the need for unwieldy single-use pattern stencils (see [3]).

**Mixed Reality** In accordance with Milgram and Kishino's proposed virtuality continuum spectrum, the term mixed reality is often used to account for most or all of the spectrum, typically excluding only the extreme of 'pure' reality itself. For example, Microsoft typically refers to its range of hardware and software products for the facilitation of virtuality continuum simulations as Windows Mixed Reality.[1] The term mixed reality will not be used in this thesis, but may be considered largely equivalent to the use of virtuality continuum.

## 1.2 Motivation

Humans interact with their environments in a plethora of ways (see ch. 2). VC applications offer great potential, opening up human-computer interaction scenarios hitherto impossible. In recent years, this potential has only grown further: Technology extends our reach far beyond our physical selves, letting us influence nearly anything that is connected to the global inter-network of hardware and software. The virtuality continuum may allow us to do this by almost seamlessly merging the real and the virtual.
In today's world, the interconnection between electronic devices is becoming denser by

[1] The (mis)use of virtuality continuum terminology for marketing purposes has a long history dating back decades (see [27] p. 1)

the day and the ubiquity of increasingly smaller electronic devices means that we are dealing with networked computers in all aspects of our daily lives. It stands to reason that this interconnected world of smart devices, together with the internet as a whole, presents an unprecedented potential for the next generation of human-computer interaction. Any user of a contemporary virtual reality simulation, for example, might have a smart assistant in their home, a phone in their pocket and numerous electronic devices around the home ranging from their laptop and router to their lights, fridge and thermostats. Most of these devices are connected with one another and/or the internet already. The virtuality continuum could allow us to tap into the way we perceive and interact with our environment's ubiquitous computing resources, integrating these interactions seamlessly into our perception.

Researchers of Human-Computer Interaction in the Virtuality Continuum need applications to conduct experiments on. When creating them, developers try to take inspiration and guidance from existing VC applications, most often in the form of games as these are currently the most common type of VC application (see ch. 4).

The simulations present on today's market however have a very narrow scope in terms of the available *objects of interaction*[2]. Most simulation systems consist of little more than a host computer running a simulation and whichever peripherals were used to display stimuli for the user (e.g. HMDs) or take input from them (e.g. cameras). The user can interact almost exclusively with the simulation itself and although sometimes the simulation will take the user's spatial environment into account (e.g. via spatial mapping), interactions with objects in the environment are few and far between.

The last decades have seen many applications and devices being integrated with a wide range of distributed devices and services in order to offer the user new ways to interact with them. Ubiquitous internet connections, gyroscopic sensors, touchscreens and vibration motors for example have changed the way in which we interact with hand-held computers like smartphones in our daily lives. It seems a logical conclusion that enabling researchers to efficiently integrate new devices and services into their VC experiments might have a similar impact on VC applications.

[2] 'Object' here refers to anything a user action may be directed at, which does not have to be a physical item.

## 1.3 Research Questions

This thesis endeavors to provide answers to the following questions:

- How can potential interaction objects be integrated into a VC system in a research context?
  - Which objects of interaction exist that could be integrated into a VC system?
  - How can these objects of interaction be categorized?
  - Which requirements need to be fulfilled for their integration?
- How can the development of VC systems with distributed interaction systems be supported by suitable development methodologies?
- Which strategies and technologies may be leveraged to achieve integrations of these objects of interaction?

Of these questions, the identification and discussion of requirements posed by the integration of new interaction objects is the most important one, constituting the center of this thesis.

## 1.4 Research Objectives

This thesis' goal is to provide a foundation for the development of distributed interaction scenarios in the virtuality continuum. It is the author's hope that in providing this, some of the challenges to researchers mentioned above may be alleviated and a path laid out for more productive and efficient development of VC applications in research contexts. To achieve this, the thesis' goal is structured into the following constituent objectives:

1. To introduce concepts unique to human-computer interaction in the virtuality continuum.
2. To analyze various potential objects of interaction and propose a catalogue of requirements for their integration into virtuality continuum applications.

3. To discuss the way in which VC applications are currently developed, in research and by the industry, endeavoring both to gauge which aspects of the development process attribute to the lack of distributed interaction and what could be changed in an effort to address this.

4. To discuss different integration strategies such as standards and technical implementations which could help address both the requirements resulting from the second goal as well as the development methodologies addressed in the third.

This thesis' goal is not the development of a one-size-fits-all solution for the integration of interaction objects. Instead, it introduces a number of experiments showcasing both the feasibility of integrating specific objects of interaction and various means of doing so. It thereby aims to highlight the *requirements* for integration solutions and to give examples for different approaches. This provides a basis for integration solutions to be developed as part of future research efforts.
In essence, this thesis is intended to be a road-map of the potentials and pitfalls which await VC researchers developing distributed interaction scenarios on their progress toward the Ultimate Display (see ch. 2).

## 1.5 Experiments and Research Contributions

The experiments mentioned in this thesis (detailed in the appendix) were qualitative pilot studies (feasibility studies). Their main purpose was to gain insight into how different systems could be integrated with one another in order to achieve distributed interaction scenarios.
The majority of the experiments conducted in this thesis have already been publicly presented: The CSTI Opening Ceremony simulation, for example, drew synergies from also serving as a public showcase of the CSTI's ongoing work during the official opening. The Shelldon project was used in a workshop as a showcase of cyber-physical systems communicating with users in a variety of ways.
The Omniscope project warrants special mention, having already been published in [46]. It entailed the creation of the eponymous multi-platform multimedia streaming and computer vision framework.

Much of the research for this thesis took place in the Creative Space for Technical Innovations (CSTI). The CSTI is an interdisciplinary research laboratory attached to the Hamburg University for Applied Sciences (HAW). Major foci of the CSTI's ongoing research are human-computer interaction, cyber-physical systems and ubiquitous computing.
A noteworthy aspect of the CSTI's ongoing research efforts into VC applications is that researchers generally endeavor to develop applications in an economical and utilitarian manner. This strategy is often described as task-oriented development or the Good Enough Approach. It refers to the applications being developed with clearly defined goals and use cases and with only minimal features intended solely to further immersion. While some of the CSTI's particulars are mentioned, especially with regards to the technological ecosystem generally used by researchers there, this thesis' results are not exclusive to this research context. The CSTI instead served only as a staging ground for its feasibility studies, during which the use of technologies unique to the CSTI was avoided almost entirely, in order to ensure relevance for other research contexts.

## 1.6 Thesis Structure

Aside from introduction and summary, this thesis is structured along its research objectives:
The second chapter corresponds with the first research goal and provides an introduction into the human-computer interaction concepts important for applications in the virtuality continuum.
The third chapter introduces a number of objects of interaction and the requirements faced by frameworks facilitating their integration.
In the fourth chapter, typical challenges during the development process for VC applications in research contexts are discussed. It also briefly introduces the concept of economical VC applications as a counter-proposal to focusing chiefly on furthering immersion.
The fifth chapter applies the requirements from the third and fourth to a number of strategies and technologies currently in use, discussing their suitability for integrating objects of integration.

# 2 Human Computer Interaction

This chapter is structured into the following sections:
Firstly, groundwork is laid by discussion human perception and perceptualization: The use of our senses and how a simulation system may address and even trick them. The common guiding concept of the 'Ultimate Display' is referenced and refined for the purposes of this thesis as the 'Ultimate Simulation'.
The second section - Interaction in Virtual Worlds - is concerned with concepts of human-computer interaction (HCI): Complex flows of action and perception, processing and reaction by both humans and machines. It is focused on those HCI aspects which are of particular relevance to VC applications.
The third section elaborates on 'physical' interaction with the VC simulation: Ways in which the user's physical perception and motion may be used to interact with the simulation.
We can communicate with other humans (and some animals) using speech. The fourth section elaborates on speech as another possible way for users to interact with the simulation.
The last section, 'Unifying Interaction Scenarios', highlights some of the challenges faced by researchers and developers while implementing interaction scenarios and proposes Polymorphic Interaction Scenarios as a means to alleviate some of them.
This chapter describes a number of devices for the facilitation of HCI. These constitute part of the means of interacting with objects of interaction. Note that the objects of this interaction themselves and their integration into the simulation system are subject of the next chapter.

## 2.1 Perceptualization

Human perception is a vast topic, spanning many research fields. An exhaustive discussion would far exceed the boundaries of this thesis. Therefore, this section contains only

a brief introduction and then concerns itself chiefly with the particulars of perceptualization in virtual worlds.
Perceptualization refers to any virtual representation of data which is made perceivable via our senses (see [20]).

### 2.1.1 "The Ultimate Display"

Sutherland's "Ultimate Display" (see [76]) is often considered the ultimate goal of human-computer interaction design. Applications of the virtuality continuum are considered to be the next step in HCI's evolution toward the Ultimate Display.
In [76], Sutherland uses the term 'display' in a wider sense to describe all computer systems designed to provide sensory stimuli to human users. Starting with describing those displays available at the time (auditory and visual), he goes on to propose ways in which their capabilities may improve in future iterations. He also suggests entirely new types of displays may be created, such as kinesthetic displays - as indeed they have been, at least to a degree.
Sutherland ends this paper describing what he calls the Ultimate Display: "The ultimate display would, of course, be a room within which the computer can control the existence of matter. A chair displayed in such a room would be good enough to sit in. Handcuffs displayed in such a room would be confining, and a bullet displayed in such a room would be fatal" (see [76], p. 2).
With this type of display no longer being confined to creating artificial stimuli but being able to perfectly (re)create a stimulus' underlying *cause*, there would be no perceivable distinction between what is 'real' and what is 'virtual'. It is arguable though that for this reason his Ultimate Display stands for an extreme which is quite distinct from any iteration of interface design which may precede it: It would mean creating a new *reality* rather than a virtual (albeit completely convincing) *facsimile*. In addition, the 'room' Sutherland mentions may prove limiting when it comes to creating the illusion of a space larger than the room. There is also no mention of the importance of *forces*: Matter alone is not enough to simulate everything. For example, there may also need to be movement, radiation (light, heat) and gravity.
For a system capable of immersing the user in an artificially created world to such a degree that they are not able to distinguish between it and reality in any way the term 'Ultimate Simulation' may therefore be better suited. It would serve as a better guide to HCI design, sharing the goal of creating convincing *illusions* rather than reality. A

definition of Illusion which serves to illustrate its meaning in this context (provided by the Merriam Webster Online Dictionary, see [49]) is: “perception of something objectively existing in such a way as to cause misinterpretation of its actual nature". The Ultimate Simulation also does not include any preconceptions about what the means of creating this impression may be. It could, therefore, taxonomically be a super-set of the Ultimate Display as envisioned by Sutherland.
A suggested model for the distinction between the Ultimate Simulation and the Ultimate Display is the difference between the Holodeck (see [47]) and Replicator (cf. [48]) concepts in Gene Roddenberry's Star Trek: The former creates perfect illusions composed of lights and force-fields, the latter synthesizes material objects like an extremely advanced additive manufacturing device (i.e.: 3D printer). The Holodeck has even found some recognition in the scientific world already, being a source of inspiration for works such as [42] and [75].
Arguably, a goal for each generation of VC system is the improvement of their capability to facilitate illusions.
Since the purpose of an illusion is to create the perception of an objective existence by misrepresenting its actual nature (cf. above), then the nature of human perception and its weaknesses to such illusions (potential for being used for perceptualizations) need to be discussed.

### 2.1.2 Senses

This section is concerned with the senses which a simulation striving towards becoming the next iteration toward its ‘Ultimate’ goal would have to be able to fool. After briefly going into a few key characteristics of each sense, the focus will be on means of displaying stimuli to these senses, both commonly available and known publicly as currently being or having been researched.
The term ‘display’ will be used to describe any technical construct designed to generate stimuli for the user. This goes somewhat against the commonly understood meaning of display, namely as a device for generating visual stimuli, but this use of display in a more generic sense is useful as an abstraction and has already established itself in a number of scientific works. Each sense will be gone into separately. It should, therefore, be noted here in advance that the nature of human perception in reality is highly multi-modal: We do not rely on a single source of stimulus when we interact with our environment. Instead, we efficiently combine many stimuli to form our understanding (see [21]). For

example, when we have a cold drink we will feel the glass smooth and cold in our hands, we will feel its weight and balance when we lift it, hear the ice cubes clinking, feel the liquid pass our lips and taste and smell its flavour. Ultimately, the right combination of different, multi-modal stimuli is necessary for making a simulation feel 'real' to us, and allow us to feel immersed and present within it. We have a keen and often intuitive understanding of the stimuli we expect to be part of any experience; the lack of any expected stimulus can have different and generally unintended side-effects. For example, if we were to grab an item but felt no mass or resistance, our ability to handle the item in an efficient manner might be impaired.
Some studies suggest that the multi-modal nature of our perception may be something we are not born with but rather develop over the course of our childhood (see [22]). This might have an impact on the design of VC simulations intended for children, as well as being a topic of research which could be explored using VC simulations.

### Visual Perception

In today's VC applications, our eyes may play the most important (certainly the most predominant) role. Visual perception refers to the way we form an image of the world by relying on our eyes as receptors for electromagnetic radiation within our visible spectrum (wavelengths of 380-740nm). An in-depth introduction into the technical processes involved is beyond the scope of this paper; for this, refer to works such as [17].
Today's visual displays generally consist of a matrix of individual light emitters which work in tight coordination to give our eyes the impression that light is being emitted by or reflected from objects and surfaces which do not, in fact, exist. A notable exception to this is the concept of electrophoresis as found in electrophoretic displays (EPDs, see appendix).
The role of visual perception in VC applications differs somewhat from its part in traditional applications such as the Windows/Icons/Menus/Pointers metaphors of desktop computers or the touchscreens of mobile devices like smartphones. Perhaps most importantly, the displays used in the head-mounted displays common in VC systems are stereoscopic, rendering slightly different images to each of the user's eyes. This can make the displayed rendering of a virtual environment seem three-dimensional, fooling a part of the user's depth perception.[1]

[1] Depth perception also consists of other mechanisms such as whether an object is only partially visible behind another. For a more detailed explanation, refer to [17].

Those applications relying on held rather than wearable visual displays, most commonly smartphone-based augmented reality applications, do not utilize stereoscopic vision.
Another aspect of visual perception is that - as part of complex multi-modal processes - it is strongly tied into our sense of balance. Unexpected and possibly even consciously imperceptible errors in the visual stimuli presented to the user, such as distortions or latencies, have been suggested as being linked to losses of balance and cyber sickness. Research into this is ongoing, with much groundwork still remaining (e.g. to measure latency in simulations). [74] [79]

### Auditory Perception

Auditory perception, too, is most often rendered in such a way as to simulate a virtual spatial environment. This not only takes into account the user's posture (i.e.: the location of their ears in virtual space, relative to the simulated origin of the sound) but also the topology and acoustics of their virtual environment. Using this information, it is possible to simulate spatial sound through a binaural headset for a user if their pose is known to the simulation.
However, the generation of accurate spatial soundscapes increases in complexity when - rather than using binaural speakers such as headsets integrated into the HMD - the user's real surroundings are supposed to carry the sound. This is often the case in multi-user applications, particularly if the required number of individually worn devices is to be reduced (possibly to zero, e.g. in a CAVE system, see [13]). Typical surround sound works well for a single user whose position is well-known, but rapidly reaches its limits in a larger space with multiple users.
A spatial sound solution attempts to simulate the acoustics of a space (such as a concert hall) by calculating and/or measuring its' expected echo responses based on the location and direction of sound sources. Techniques such as Wave-Field Synthesis (WFS) then attempt to mimic the exact soundscape the room would display within a section thereof. A recent example of how this can be achieved by leveraging a GPU can be found in [4]. While WFS is computationally very expensive, the fact that a VC simulation will likely need at least one powerful CPU in any case could be a synergy.

### Haptic Perception

Humans can perceive objects and events in physical contact with their bodies in a variety of ways. A suggested subdivision of haptic sensations is that of cutaneous and kinesthetic perception.
Cutaneous perception refers to tactile sensations originating in our skin. The texture and heat of a surface in contact with our skin would be detected by our cutaneous system. Kinesthetic perception on the other hand focuses on our muscles, joints and tendons, where mechanoreceptors allow us to gauge the weight, mass and speed of an entity we are touching. [39] provides a comprehensive introduction into the concepts of human haptics and forms the basis for discussion of that field in this thesis. For a discussion of the significance of haptic sensations in the design of multimedia interfaces, in general as well as for virtuality applications, refer to [68].
While they were rare in early VC simulations, there is ever-increasing support for the generation of haptic sensations. They tend to be focused on providing stimuli to cutaneous receptors through tactile feedback channels. In the construction of basic tactile interfaces, the most commonly seen mode of generating stimuli is vibration motors. The VibeGlove experiment demonstrates the use of small coin-cell size vibration motors (such as those used in cellphones) for tactile feedback in a VC application. Such vibration-based systems are extremely common in phones, console game controllers and VC interaction devices.
The exploration of an object using touch (rather than vision and audition) may use tactile feedback extensively but is not confined to it, also including kinesthetic aspects such as the amount of force required to move the object or transform its surface. Though it has been demonstrated that tactile and kinesthetic feedback can be substituted to a degree (see [65]), the ability to provide fine kinesthetic stimuli to our bodies' internal mechanoreceptors is still in its early stages of research and development.

### Other senses

Some experiments, showcases and studies exist as to the perceptualization of other senses in the VC. In [12] and [35] for example, olfactory displays are presented. Gustatory displays have also been proposed, though many rely on the multi-modal nature of gustation: In [55] for example, Narumi et al. showed how a pseudo-gustatory display could work by simulating the look and smell of an eadible item during its consumption. In Augmented

Gustation Using Electricity (see [54]), Nakamura and Miyashita even suggested direct electrical stimulation to the tongue in order to achieve 'electric taste'.
Displays for such senses have not reached widespread adoption like the ones mentioned above, but may still be of interest in research contexts.

### 2.1.3 Presence

Presence is a concept which is of particular interest in VC use cases. A commonly cited and well-established definition can be found in [85]:
"Presence is defined as the subjective experience of being in one place or environment, even when one is physically situated in another. [...] As applied to a virtual environment (VE), presence refers to experiencing the computer-generated environment rather than the actual physical locale."
They then go on to explain and disambiguate involvement, immersion and presence. Both involvement and immersion are established as necessary conditions for presence.
Presence (as well as its conditions) is often seen as a measure of a virtuality application's quality. This may be challenged (such as when an application's purpose does not require presence), but for many applications it is clearly an important and desirable feature.
Immersion is often considered to be directly related to how 'realistic' a simulation is - in how far the simulation mimics reality. One implication of this is that the most easily accepted interactions (those conducive to presence) are ones natural to humans.

## 2.2 Interaction in virtual worlds

In this paper, interaction refers to human computer interaction. From a technical perspective, it can be characterized as a sequence of actions which the user and the simulation system perform with both a goal and an effect on the other.
Virtual worlds - also commonly referred to as virtual environments (VE) [85] - form the basis for all human-computer interaction in the virtuality continuum, setting a stage for interactions to be performed in.
The interaction types explained in this section are structured by the way in which we may provide input to the virtuality system with recognizable actions; the corresponding feedback channels will be mentioned throughout.

From an architectural and technical point of view, the facilitation of interactions in virtuality can be structured into the following constituent parts:

1. Input: Reception of user input
2. Processing: recognition and quantification of intended meaning, mapping of intent to effect
3. Output
    a) Feedback/System Reaction: Output in **reaction** to processed events, such as feedback upon collision of real and virtual objects
    b) Unsolicited output/System Action: Any output stimuli other than designated feedback. Especially includes those stimuli which are intended to initiate an (unsolicited) interaction with the user. They are the VC system's equivalent to a user's **action**.

### 2.2.1 Input

A user's input to a VC system first has to be received at all. This is achieved using a multitude of sensors, such as cameras and buttons.

#### Tracking

Tracking systems in the context of the VC are combined hardware and software systems which collect and process sensor input to gain information about the state of real objects (position, orientation, speed, trajectory). Most commonly, they consist of specialized cameras positioned and oriented relative to one another to allow for the interpolation of position and orientation of anyhting seen by several of them. Many other techniques exist however, such as precisely timed line laser sweeps (such as the one used by the HTC Vive's 'Lighthouse' tracking system, see [38]) or the measurement of signal flight time (such as for Global Navigation Satellite Systems (GNSS)). [45]

### 2.2.2 Processing

The input is then interpreted, assigning meaning to the input. The processing of input is where user input is analyzed and (possibly) an intent is established. Depending on the input system, that can be trivial (e.g. traditional keyboard) or highly complex (e.g. multi-camera visual system for gesture recognition).
Processing can be divided further into a variety of individual processing stages. For example, a tracking system may use various sensor readings to triangulate positions (e.g. those of a user's fingers) in real space. Then positions in real space are transformed into a position in simulation space. Finally, a range of positions and orientations over time are analyzed to form an intent - for example: The user was pointing at a specific virtual entity.
Once an intent is established, it can then be used by the simulation to figure out an appropriate effect and response. For example, a successfully interpreted pointing gesture could result in a switch to an editing mode for the relevant mode (allowing the user to then pick a color or resize or move it). It could also result in various feedback channels to the user being activated, e.g.: A sound could be played, or the object could be highlighted. This concerns the next technical stage of interaction: Output.

### 2.2.3 Output

The output stage concerns orchestrated output in the form of various stimuli, either in reaction to the original events or as an action initiating a new interaction from the system's side. The output is where displays (in the wider sense) come in: The system acts in a way perceptible to the user.
Output channels can be categorized into feedback in reaction to user actions and system actions unsolicited by the user. In many cases, the output stage is triggered as the result of the system having processed a user input. This will henceforth be referred to as 'feedback' of the VC system. In some cases, however, the system output may not have been triggered (intentionally or unintentionally) by the user, but instead by the VC system's internal processes. The term 'unsolicited system action' will be used for this.

### Feedback/System Reaction

One form of output from the simulation system is that of feedback, i.e.: output generated after processing user input in order to signal a reaction to it. Feedback is important for a number of reasons. Humans expect certain feedback from their environment and rely on it to fine-tune control loops in their interactions. When we, for example, lift an object in order to move it, we rely on our senses to tell us about its temperature, surface properties, weight/mass (via kinesthetic senses) and so on in order to react accordingly - be that to drop it if it is hot, grip more tightly if it is slippery or use both hands if it is heavy. When interacting with an environment that is - at least in part - virtual, it is often up to the application to provide this kind of sensory feedback.
Aside from adding to immersion by simulating environmental and object properties, sensory feedback may be used for other purposes. For example, if the user were to ask a question (verbal interaction, see below), feedback may include the answer itself (be that in the form of a visual display, the manifestation of the result of an interpreted command or an auditory response). It may also simply indicate to the user that input was received by the system at all. This is particularly important if the desired outcome is not immediately noticeable, for example if off-site speech-to-text (STT) processing takes a moment to return a result or the interpreted command results in an action that may not be observed (such as sending a message or changing a system setting). Another possibility is that the application failed to interpret an intent with a reasonable degree of certainty - it is important for a user to know this, so they do not expect a result and can try again as required.
Feedback could also be used to convey more abstract sensations like the presence of an imminent threat. An example for thermal feedback to this end can be found in [41].
Finally, feedback may also include reactions of the system to non-user actions and events. For example, if another person (not subject to the simulation) were to step too close to a user whose sense of reality is impeded, such as by wearing an HMD, it may be sensible to warn the user and/or the other person of a possible accident.

### Unsolicited Output/System action

Some system outputs may not be prompted by human action at all. The system may also need to display to users other unsolicited information, such as status information about the system or the user's environment. In some cases, the system may even initiate

an explicit interaction with the user, prompting action on their part. For example, if a constituent device of the VC system were to malfunction, lose connection or run out of battery, the system may display a warning to the user. This could have purely informative character or be intended to trigger an interaction with the user. For example, the user may be informed of a low battery status of an interaction device such as a wand (not prompting immediate action) or need to handle a prompt asking whether they want to switch batteries right away, use a different input device or end the simulation entirely.

## 2.3 Physical and Manual Interaction

We interact with our environment in a variety of ways. Among the most important is the use of our extremities (our hands and feet in particular), generally in combination with feedback via our senses.

**Tracking of the user's physical motion** Motion tracking systems, probably the predominant type of tracking system for immersive VC simulations, obtain information about the exact position, pose and trajectory of the user's body, body parts or tools. These are then transformed from real space (coordinates in reality) to simulation space (coordinates within the simulation). This allows the user's actions in reality to be processed as relative to simulated physical objects in the virtual environment, allowing for an immersive melding of the two planes.
Despite the widespread use of a wielded or carried item whose position and rotation is tracked by the simulation in order to extrapolate the position and orientation of extremities is often considered to be inferior to tracking of the user's body itself (see [40]).

### 2.3.1 Posture and Gestures

Posture refers to any static, observable state of a human's body, in particular with regards to how their outward appearance is dictated by the transformation of their skeletal structure by means of their musculature. A posture may be characteristic for a person or assumed for a specific purpose, such as conveying meaning (e.g. a hostile attitude) to another.
Posture generally does not lend itself to any immediate feedback channel other than visual via the change of perspective the user has on the scene. Due to the difficulties of

tracking and rendering the user's entire body pose as well as providing an angle to allow them to view it, showing them their own pose is rarely done (though possible, e.g. using full-body motion tracking systems like the ART system and simulated mirrors).

Gesture can refer to any movement or change in posture which a human might perform. We use them in many different scenarios: to convey messages to one another, in rituals, as mannerisms, even unwittingly or unwillingly as reflexes[2]. Hands in particular are key to a great number of ways we interact with the world, perhaps more so than even our feet or voices.
Gestures also do not generally lend themselves to any feedback channel other than visual, though showing the user's extremities is easier in this case than for postures (where even more of the body is generally required). Both Space Flight and the CSTI Opening scene provide examples of this by showing a representation of the user's hand, captured by the LeapMotion hand tracking system and rendered in the scene.

In this definition, Postures and Gestures do not generally involve the manipulation of a physical object. At most, such objects might be involved in a supporting, static and/or passive role, such as a wall which the user may lean against (a posture) or a glove on the hand that the user waves with (a gesture). For other uses of objects see section 'Manipulation' below.
Postures and Gestures are among the most important aspects of the user's actions for a VC application to capture and analyze. They can be leveraged to permit the user to influence a VC application in any number of ways without requiring the direct manipulation of a tool.
Research into which gesture is best suited for which task is ongoing and it seems unlikely that it will reach a clear conclusion in any case. According to the limited research available, such as [34], Users also seem to have individual preferences as to which gesture they would rather use for which interaction. The aim of studies such as [34] is to find out which gestures are commonly preferred for each interaction in order to accommodate at least a majority of the users. Since they only allow the users to choose from a predefined set of gestures, methodologies like the one employed in [40] might be more appropriate. Here, Leng et al. let test subjects design their own gestures and tried to find out which gestures could be most universally agreed upon.

[2]Though most gestures that are of importance in VC applications assume that they the user executes them on purpose.

### 2.3.2 Manipulation

Humans are proficient at handling tools, usually as an extension of their own bodies. In an extension of how we - as it were - *manually* interact with the world, the use of tools in our daily lives is often ingrained from an early age. The primary mode of manipulating objects is the use of our hands. Other modes are possible, such as with our feet when we kick a ball around, or with our shoulders and torso when we have our hands full and shoulder a door open. While the term manipulation originally explicitly refers to using our hands (lat. *manus*: hand), in this paper, the term manipulation will be used for all interactions which allow us to change the state (position, rotation) of another object using our body to touch and potentially move it. Manipulation is still to be seen as a form of gesture, constituting a subset of gestures in general.
Manipulation gestures are also easily conflated with the use of interaction tools such as wands. In this thesis, it is assumed that the object of a user's 'manipulation' is always a virtual one unless explicitly stated otherwise. The use of a physical object as part of an interaction is a different issue. Most often this fits into the topic of motion tracking instead. For example, a simulation may track a wand rather than directly tracking the user's hand - so long as the user is holding it, the hand's position can be inferred.
Typical feedback channels used in manipulation scenarios are tactile/haptic, audio and visual feedback. For example: In the CSTI Opening scene, if a purely virtual wall fragment object is shoved by the user, they can see the object moving and hear it collide with the hand as well as the floor and other wall fragments. Using a tactile feedback device such as the vibe glove would add a tactile feedback channel.

### 2.3.3 Virtual Objects: Gesture vs. Manipulation

When interacting with virtual objects, the distinction between manipulation and other gestures can become somewhat blurry. For example, imagine a virtual ball displayed to the user, hovering at chest height in a VR simulation - a setting incorporated into the SpaceFlight simulation. If the user were able to see their hands (e.g. via an integrated motion tracker, see VibeGlove experiment) and shoved the ball using their hand, this would clearly constitute a direct manipulating gesture. If the ball were to be moved - somewhat more indirectly - through another interaction such as a gesture with a wand as input device, this may be less clear: 'Magical' gestures which do not follow patterns

that in reality would result in a causal link between action and reaction are difficult to categorize.

## 2.4 Speech - Verbal Interaction

When it comes to conveying meaning to another human being (and even to some animals), we often use our voices.
In contemporary VC applications, verbal interaction is still relatively uncommon. It seems a likely assumption that this will change soon: The recent years have seen a rapid rise in popularity of verbal interaction scenarios fueled by the universal adoption of speech input on smartphones and the popularity of so-called Smart Speakers/Assistants, e.g. Amazon's Alexa system. These verbal interaction scenarios will likely form the basis of many future interaction techniques in the VC.
While this section will introduce the established ways in which these verbal interaction scenarios present themselves to the user, note that many of the technical aspects of their implementation are within the purview of the next chapter.
Though speech is an important facet of our communication, we do not generally rely on verbal communication exclusively, instead supplementing it with gestures, facial expressions and other means intended for visual and possibly tactile recognition by our counterparts. This should not be neglected in virtuality systems and verbal interaction systems should use voice recognition and transmission in unison with other features such as facial expression recognition where ever possible.

### 2.4.1 Contemporary Verbal Interaction Systems: Smart Assistants

Most verbal interaction scenarios these days require no non-verbal user action for their initiation. When the user wishes to initiate an interaction, they utter a 'wakeword': A phrase which indicates to the system that the following words are intended for it to react to (e.g. 'Hey Mycroft!', 'Alexa' or 'OK Google').
Since the wakeword is used akin to the name of a person the user is trying to address directly, it is often used to identify the smart assistant system they are addressing (e.g. Amazon's Alexa). Companies are linking this directly to their brand - likely because the humanisation that comes with 'being' the helpful smart assistant is good for tying the

customer to their brand, increasing loyalty and making them less likely to switch to a competitor.
The sentence following the wakeword is typically either an imperatively phrased command to the system (e.g. 'Set a new alarm for 6 am!') or a question ('What is the circumference of the moon?'). Ideally, no special syntax is required and the user can use natural speech. In reality, the systems' ability to process the sentence in order to understand the user's intent and react accordingly generally corresponds with the clarity and simplicity of the sentence structure. As a result, commands and questions are typically phrased in a brief and simple manner.
If the processing of this input leads to a recognized intent, the system responds. On mobile devices, this typically includes a recognizable 'success' feedback, such as a short vibration or a distinctive sound as well as the execution of the required action or the display of the answer to the asked question. Smart speakers tend to rely on audible responses, though screens are becoming increasingly common (e.g. in the newest generations of Amazon's 'Echo' line of smart speakers).
If no intent could be interpreted within a reasonable margin of confidence, the system may ask for clarification ('Sorry, I did not understand that!') or attempt to delegate finding a suitable intent to another system. An example for this would be the Mycroft system which allows for the registration of 'Fallback skills' such as inputting the results of the Speech-to-Text processing into the WolframAlpha engine for further analysis ([53], [86]).

### 2.4.2 Verbal interaction in the VC

In terms of how commands and questions are phrased, verbal interaction in the VC would not have to seem very different to the user compared to the use of any smart assistant. From wakewords to the phrasing of commands and questions, the interaction could stay much the same. There are primarily a number of technical differences and challenges (discussed in the next chapter). The output of the VC system however differs significantly from the usual smart assistants. The VC system can alter the user's perception of their environment much more profoundly than any smart assistant. The user could shape the virtual world to their will using words in ways not possible in reality - comparisons to magic and biblical interactions come to mind. 'Fiat Lux' could make the sun shine, asking the VC system for the location of an entity inside the simulation could result in it being shown through walls and other visual obstacles, commanding it to create a new

entity could make that entity appear.
The use of verbal interaction in such a context has been explored by various scholars, novelists and screenwriters for decades (once again the Holodeck springs to mind, see [47]), though it remains to be seen whether and in which form it will become a widespread standard for complex interactions in the VC.

## 2.5 Realism tradeoffs

For decades, the development of user interfaces and interaction techniques has been contrasted with the increase in power and efficiency of computers (cf. Moore's Law). Some researchers have recognized that the design of user interaction techniques did not change constantly but rather showed periods of almost complete stagnation followed by rapid change. In [32], Jacob et al. suggested that the development of user interaction techniques might lead them to become more progressively more 'real' and proposed a framework for reality-based interaction (RBI) as early as 2008. Forward a decade and virtual, augmented and mixed reality has become nearly commonplace - reality-based interaction had indeed increased significantly. As they pointed out, however, this change would come with tradeoffs:
"[...] there are times when RBI principles should be traded against other considerations. We propose that the goal is to give up reality only explicitly and only in return for other desired qualities, such as:

- Expressive Power: i.e., users can perform a variety of tasks within the application domain
- Efficiency: users can perform a task rapidly
- Versatility: users can perform many tasks from different application domains
- Ergonomics: users can perform a task without physical injury or fatigue
- Accessibility: users with a variety of abilities can perform a task
- Practicality: the system is practical to develop and produce" [32]

Their proposed tradeoffs apply to many aspects of the development of a VC application and will be mentioned throughout this thesis.

## 2.6 Unifying Interaction Scenarios

Interactions within the VC take many forms, including among others the ones mentioned above. For every intent the user might have - like the selection of a virtual entity - there are any number of interaction scenarios which may be employed.The user might isolate the entity with a wand or by simply pointing at it with their finger, they might vocalize the name of the entity, verbally describe its position or even simply look at it.
Each of these approaches would necessitate very different implementations and not all of them may even be available or suitable for the simulation or the user - especially if they depend on hardware that will not be available to the user, or if the user is handicapped. It would be helpful to researchers creating VC simulations if there was some way of describing the intent once and then enabling different means of conveying it to be easily substituted, possibly even during runtime. If a simulation were built to use wands as a means of interacting with the environment, how could they be replaced with a hand tracking system? Which gesture would equate which action?
Such substitutions might be possible if the interaction scenarios were based on a common standardized model. This would open the system to configurable interaction setups. If one were to categorize the possible intents of interaction in a way that could be (more or less) universally agreed upon, the integration of new interaction systems could be much simpler. To some degree, this has already been proposed: [17], for example, categorized interactions into ‘selection’, ‘manipulation’, ‘navigation’ and ‘system control’.
Similarly, even the input may be disassembled and categorized. For example, the input system could be defined as providing one or more primitive values such as boolean values, floating point vectors, quaternions and character strings.
This thesis proposes the development of Polymorphic Interaction Scenarios as a generic approach for the description of interaction scenarios which would allow flexible mapping of input data to an intent.

### 2.6.1 Polymorphic Interaction Scenarios

Polymorphic Interaction Scenarios (PIS) are characterized by the goal of an interaction, with different argument sets being supported to achieve it. At the center of polymorphic interaction scenarios is the concept of re-use. The overloading of method signatures in programming languages like Java or C# can serve as an example for how this can be achieved.

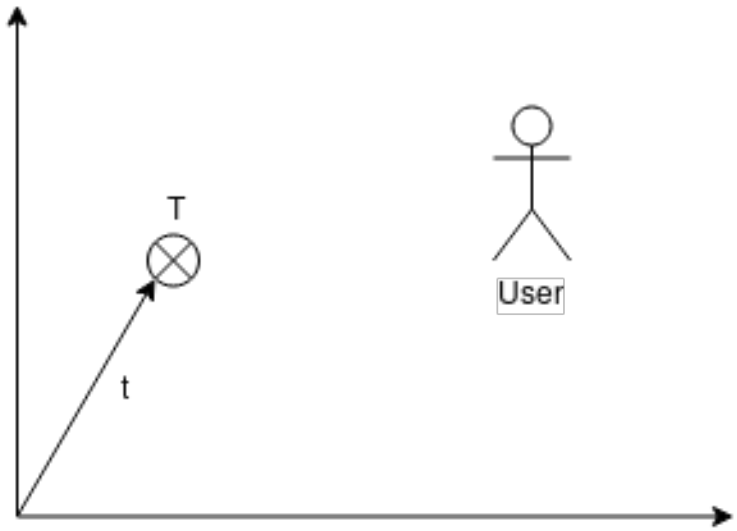



Fig. 2.1: Selection of target entity T by position t in world space

For example, the interaction scenario of selecting an entity (visible object) in world space could be defined as the mapping of a character string source (which could be used to input an entity's unique identifier or name), one vector (indicating the entity's position) or two vectors (indicating the origin of a selection gesture as well as a direction towards the entity).
A more generic approach to describing and mapping actions and interactions could go a long way toward maintainable, economical and future-proof VC applications. It may even make it possible to integrate new subsystems into a simulation without having to change much of the underlying interaction design. For example, the integration of a speech recognition system for selecting entities could consist of adding only a way to turn sound buffers containing the user's utterance into a character string - which can then be used with the existing method for selecting the entity based on its ID or name. Polymorphic interaction scenarios are in line with the concept of economical VC applications (see ch. 4).
Illustrations 2.1, 2.2 and 2.3 demonstrate how the same goal (selection of target entity T) can be achieved in at least three different ways depending on available arguments.

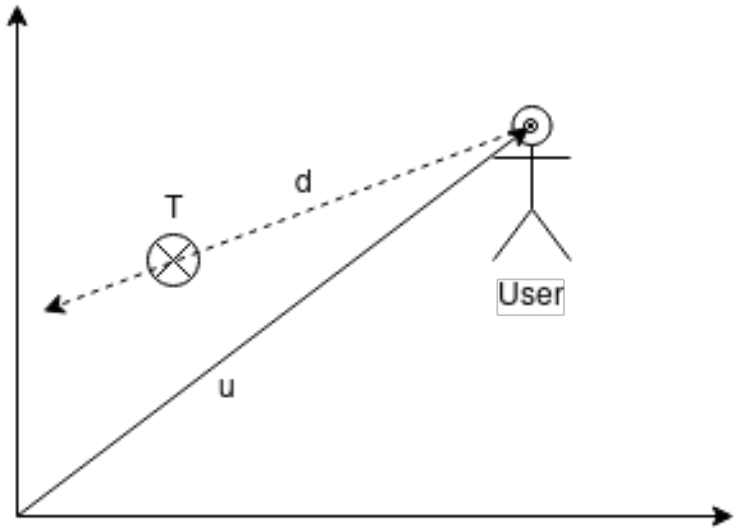



Fig. 2.2: Selection of target entity T by gaze tracking: Ray-cast vector d (gaze direction) from origin as described by vector u (world space position of user's head-mounted display)

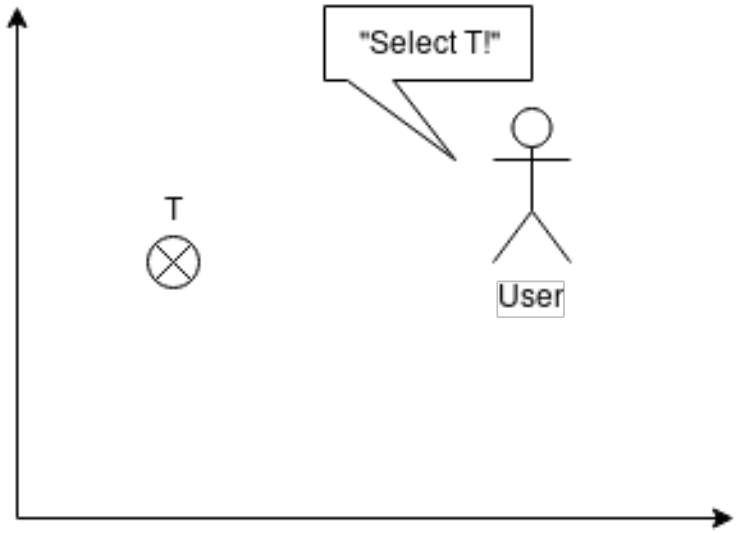



Fig. 2.3: Selection of target object T by name using speech recognition

# 3 Objects of Interaction

Virtuality applications are increasingly being recognized as having the potential to offer a wider spectrum of use cases beyond those simulations which dominate the consumer market at this time. To offer new use cases and new interactions to their users, they will often require additional objects for the user to interact with, both real and virtual. Note that *object* in this context refers to the object of an interaction scenario (as opposed to the subject, which is the user) and not necessarily a physical item or entity.
Adding objects of interaction would require the simulation system to be integrated with other systems. System integration in general is well-established term and encompasses significantly more than just the technical combination of constituent systems to achieve a unity of function and purpose. However this thesis will focus primarily on the technical requirements of such an integration and only within the VC.
This chapter aims to shine a light on various objects (devices, systems and services), highlighting the opportunities and challenges their integration may present. In addition, it offers a taxonomic base and classification. It also lists a number of challenges and approaches to the networking of virtuality system components. Though it is certainly not a distinction without some degree of intersection, this discussion is structured into the following segments:

- First, Virtual Periphery deals with software-based aspects like accessing programs on the simulation's host computer or other resources through networks.
- Secondly, Multimedia and Streaming provides insights into the often particularly challenging tasks of multimedia streaming and performing analyses and transformations on audio and video.
- Thirdly, the peculiarities of cyber physical systems and smart environments will be discussed, especially concerning how they interrelate with AR/MR systems.

- Fourthly, integrations of the user's physical environment will be expanded upon by discussing the special case of integrating 'dumb' objects, i.e. physical items that are not directly connected to a VC system.

### 3.0.1 Categories of Requirements for Integration

In the study of information systems, there are customary approaches to the categorization of requirements. An example for this is the juxtaposition of functional and non-functional requirements. Since this thesis deals only with the requirements of system integrations within the VC, however, a categorization tailored specifically to this type of use case may serve to better structure the debate. The following proposition for such a categorization will form a taxonomic cornerstone of the discussions in this thesis:

- Access Conditions
- Communication Requirements
- Interaction Requirements

This thesis focuses explicitly on those requirements which arise from the nature of VC systems as opposed to generalized information systems: It prioritizes the discussion of problems specific to the integration of systems into a VC system. It is not the goal to illuminate all possible requirements posed by the integration of systems into an information system in general.

**Access conditions** concern the conditions under which the simulation and the object may access each other's services and data. This particularly includes a number of non-technical requirements relating for example to legality, privacy and security. The integration may for instance require the researchers to agree and adhere to contractual obligations for access (such as payments).

**Communication requirements** are requirements caused by the need for communication between the integrated system and the simulation. Most non-functional requirements such as Quality-of-Service (QoS) requirements are part of this category. The need for well-documented data interchange formats would also belong into this category.

**Interaction requirements** arise out of a need for interaction between the user and the system. This often includes the need for new interaction techniques which allow the user to utilize the potential added by a newly integrated interaction object. For example, there may need to be a way for them to perceptualize the integrated system.

### 3.0.2 Ethics and Privacy

A VC application can include the collection, transmission and processing of significant volumes of sensitive data. The use of distributed interaction systems only exacerbates this, as data may be exchanged between systems in different networks and under the purview of different maintainers and operators. These data interchange processes are subject not only to local and international law but also to ethical concerns. For example: As mentioned throughout this thesis, cameras and microphones are used ubiquitously, especially for motion tracking and object and speech recognition. The data they produce is generally of an extremely personal nature and its transmission, especially to third-party service providers, should be recognized as a potentially risky proposition necessitating ethical considerations. This is especially true if a breach of security resulting in a data leak cannot be ruled out with a satisfactory degree of certainty - though even the use of data within license agreements can be problematic.
An example for this which has by now reached a certain level of notoriety is Amazon's Alexa smart assistant system based on local 'Echo' devices. These devices have even been found to have the ability to listen to conversations in adjacent rooms, making them a privacy risk not only to the immediate, controlled locale of the VC system but potentially even to people outside of it. Controversies have included the distribution of Alexa's recordings to Amazon's service and development personnel and even the leak of private recordings to unauthorized third parties. Such problems are hardly exclusive to the firm though - Google for example has been discovered treating some of their smart assistant system's recordings in similar ways and making them accessible to employees. [5], [83]
Another example for problematic handling of user data can be found in the services offered under the 'Vuforia' label by PTC Inc. The services - a wide variety of technologies to help with the development of augmented reality applications - consist of both local software packages and cloud-based services. In particular, one implementation of their service for recognizing two-dimensional objects (images) in a live video requires uploading the 'Image Target' to their cloud servers for processing and (optionally) persistent storage. Lack of transparency about where which data is processed and saved gives rise

to privacy concerns.
It is also important that each aspect of a distributed interaction system be given ethical consideration aside from privacy-related issues. The effect on various demographic modalities such as gender, race or age on the accuracy of biometric identification systems has seen some scrutiny by the scientific community. In [60] and [63] for example, algorithms which were already in use by governments were analyzed, with the results showing demographic biases for a number of reasons. In order to avoid situations in which the VC system may show unfair biases towards or against certain demographics, these concerns should factor in the researchers' considerations when integrating systems and algorithms with a potential for biases. It is also always advisable to test VC systems with a wide enough range of different demographics to catch such problems early.
If the assurances of service providers conform with the research laboratory's ethical requirements, the rights of individual users to make their own decisions about the use of their data still remain another factor to be considered. Any risks to a violation of their privacy should be noted in the VC application's own terms of use and the conditions of a trial and discussed openly with users. This especially applies to test subjects who may not be aware of the research laboratory's standard practices.

## 3.1 Virtual Periphery: Software Integrations

Virtual periphery concerns software systems which may be integrated with a VC application. Virtual in this context means that the periphery's hardware specifics are largely irrelevant. The user does not interact with hardware components of this object in any other way than through virtual interactions offered by the VC system. It does, for example, include the integration of a web-based service or a local file system. It would not include the integration of a user's smart speaker, since they may interact with it physically as well.
Virtual workspaces are of particular interest in this section. These are simulations and applications which fall within the VC and are intended to offer an alternative to traditional workspaces, such as setups based on the desktop metaphor.
Throughout this section, software systems and components will be described in relation to a number of largely orthogonal traits. The most important of these are:

- their locality, e.g.: whether they are run on-site or hosted elsewhere

- their coupling, e.g.: closely coupled code modules or loosely coupled standalone applications
- their code license, e.g.: ownership, open source license or closed source/proprietary license

Structuring the discussion along these traits allows specific challenges and requirements to be highlighted from different perspectives.

**Realism tradeoffs for software integrations** Any software application may contain objects of interaction whose existence is entirely virtual: There may be no realistic way of portraying them inside a simulation. Many office workers in particular interact with (almost) entirely virtual business functions. The abstraction from physical reality is part of the functions they perform. For example, companies often rely on enterprise resource planning (ERP) systems for many tasks such as accounting, reporting and fiscal analyses. The concepts and objects the users interact with in these systems - such as a sales prognosis - often do not have any physical form which the user may interact with, rather they exist only virtually. As such, a realistic representation of entities and tasks is often not a guiding principle that can be followed. Instead, the simulation needs to offer other - metaphorical - ways of illustrating and manifesting the meaning and function of these virtual objects. Many such metaphors have been previously developed for traditional computer workspaces. They could be seen as an example of a realism tradeoff as proposed by Jacob et al. in [32].
In order for such virtual interaction objects to be offered within a VC application, their representation may have to be adapted. The research and development of suitable representations inside the virtuality continuum is therefore an important prerequisite for the integration of many such virtual interaction objects.
In addition to the manifestation of the object of interaction, even that of the user itself, their perspective and available (inter-) actions may be subject to similar considerations. It may solidify immersion if the user were to sit in front of a virtual desk with virtual papers on it, but that well be counterproductive and indeed offer little benefit over traditional computer-based interactions with their business computing systems. Instead, sacrificing some immersion and realism for more extensive use of the VC's benefits over traditional interaction systems may lead to increased productivity. A data analyst for example may profit more than anything else from using the vastly increased field of vision

(compared to monitors) and depth perception when browsing data and creating illuminating visualizations. This is exemplified in simulations such as those described in [15] and [2]. These indicate that a more 'immersive' and natural interaction may sometimes be counterproductive.
The research of benefits and drawbacks to these approaches and the trade-offs between realism and productivity is ongoing. Particularly the concept of creating task-oriented simulations foregoing the pursuit of immersion as a goal for its' own sake is a topic of research in the CSTI and is further elaborated upon in ch. 4.

### 3.1.1 Local Software

Probably the most important type of system to interconnect with a simulation is its own host system and local computer network.
A VC application may be required to interact with its own host system. For example, it may need to be able to look for, show and modify files on the local file system. Examples for this include not only productivity applications such as office products (text editors, spreadsheets etc) but also the editing and cataloguing of photos and videos. Thanks to well-established software libraries, access to its host's systems is not unusually problematic for a VC simulation compared to any other application. In some cases such as browser-based VC applications (see below) the sand-boxed nature of the application may deny it access to some systems by design, but there is little difference compared to any other web-based application. The primary challenge here is the design of suitable interaction techniques and metaphors.
Standalone applications sometimes lack efficient ways to communicate with other local applications. Though operating systems offer communication services like message buses (e.g. the Linux D-BUS IPC system), these often go unused. Many applications offer few - if any - means for communication with other applications. If they do, these can range from efficient to ineffective. Even inter-systems communication via the file system is not unheard of - which can be extremely slow and prone to many problems such as inefficient hardware access, lack of transactional processing and unexpected file system privileges. A typical way for standalone programs to communicate locally is through web interfaces (such as SOAP or REST over HTTP), which has the additional benefit of making them easy to expose to both local networks and the internet.
In the case of software modules integrated directly into the simulation, establishing communication is often less of a problem, although some difficulties may arise from having to

adapt code and data structures between different programming languages and systems. For an example, refer to the Omniscope project's exchange of data between C, C++ and C#, especially using C-style bindings.
Researchers of VC applications often need access to those systems which are of particular importance to the core of the simulation itself: The host system's graphics processing units (GPUs) in particular offer many features which could be used to great benefit inside the simulation, such as programmable shaders for fast parallel processing. Modern IDEs and game/simulation engines offset some of the technical challenges of this integration, offering complete rendering pipelines and extensive integration with the host's systems already. As such, the integration of host systems becomes challenging mostly if researchers need the simulation to work in a way unintended and unprovided for by the IDE's or simulation engine's developers. In this case, it would be necessary to gain more immediate access to the relevant systems in order to implement them. Simulation engines do expose access to some of their host's systems via APIs, but these may not be sufficient depending on what is trying to be achieved: Their APIs tend to be geared towards commercial applications (games, in particular); research contexts often have different requirements (see [46]).
A particularly challenging use case calling for extensive access to the host system's resources would the custom development of an entire simulation engine, but due to the extreme complexity and cost this is a relatively rare endeavor. It will therefore not be expanded upon in this context. It may be sensible to refer to open source frameworks like A-Frame [1] and rendering engines such as OGRE [57] for examples and possibly a springboard for development.
Devices and services on the local network are also fairly easy to connect a simulation to. Mostly these do not add many additional difficulties specific to VC applications. Note that many locally networked devices of particular interest for integration into a VC application are cyber-physical devices, especially those comprising a 'smart environment' or 'smart home' - these are discussed in their own, separate section.
This category of services is also of particular importance for many productivity applications. Applications used in business and industry most often run on workstations with desktop style interfaces or on the touchscreens of mobile devices. If a VC simulation were to serve as an work environment for a job which typically relies on a smartphone, desktop or laptop computer, a number of challenges would have to be overcome. Firstly, the user's interactions are quite different from the traditional desktop environments. Secondly, the user would ideally be able to interact seamlessly with applications intended for VR and traditional desktop systems (i.e.: desktop style applications) which have not

been adapted yet. In lieu of a legacy application or an existing VC adaptation thereof, there is also often the option of creating new software: The benefits and drawbacks of building a custom solution vs. (re-)use of an existing one under license will also be discussed.
Note that the term 'legacy' will be used to refer to any design, concept or technology which is not originally intended for use with VC applications and whose characteristics may make using them in conjunction with VC applications difficult.

**Supporting Legacy Interactions in the VC** While some programs may be somewhat compatible with integration into the simulation itself (perhaps because they run entirely 'headlessly', i.e. without any graphical user interface, in any case), others may have been intended for use in a legacy setting, e.g. a desktop UI or a smartphone's touch screen and sensors. Users are often very intimately familiar with their use, making the switch to a completely new type of interaction challenging and causing them to be reluctant to switch at all. The development of a VC-compatible version of such programs therefore presents a challenge. This section deals with accommodating the possible need to make concessions during this development in order to support legacy interactions, in accordance with the realism tradeoffs proposed in [32].
The switch from traditional to VC interaction techniques is the latest addition to a long struggle to understand and influence the process of user interface revolution. For example, while it has been established that the QWERTY keyboard layout has a number of drawbacks and although many alternatives have been suggested, it remains the most commonly used keyboard layout (give or take a few differences due to internationalization). The ability to change the pre-assigned keyboard layout post-production, a suggestion put forth by Cummings in 1984 (see [14]), based on Noyes' review of the layout a years prior, in [56] has become ubiquitous: Touchscreens as opposed to traditional keyboards do not even have characters printed on them. Changing the layout is trivial. Nonetheless, the QWERTY layout remains steadfast. The changes required by applications in the VC are even more drastic, e.g.: switching from using two to three dimensions, from mice to wands and from keyboards to speech recognition. Further research into how best to achieve revolutions in technological adoption in general and user interaction systems in the VC in particular will be required.
Legacy UI designs such as desktops or smartphone UIs rely on a fairly limited amount of interaction types which can find all manner of expression in the simulation. For example, two-dimensional frames in virtual world space may be used to emulate typical desktop

interactions, together with touch interactions for selection. Since touch interfaces (touch screens) have become more and more commonplace these days, an ever increasing number if applications (including the major operating systems) offer support for touch gestures, which can be implemented in the VC with relative ease. An example for this can be found in [73]: Their team compared two types of mid-air touch gestures inside a VC simulation for interacting with a two-dimensional menu.
Research concerning the feasibility and suitability of a variety of interaction types is ongoing, many of these studies even attempt a direct comparison with legacy interactions. The many types of interaction and comparisons necessary are leading to the creation of a great number of VC applications. Polymorphic interaction scenarios as introduced in ch. 2 may prove a valuable tool for developing such applications faster, more efficiently and with a focus on re-use. They may also lead to more comparability between scenarios.
There are a number of attempts at what could be called intermediary interaction designs which utilize aspects of both traditional interaction designs and new concepts developed for the VC. Microsoft's current 'Windows Mixed Reality' offerings (available for their HoloLens and other designated Windows Mixed Reality (WMR) HMD based systems) are examples for this. They appear to be intended to accommodate VC applications and introduce users to new interaction scenarios while at the same time not straying far from the desktop style of their Windows operating system. The result is sometimes an unintuitive mixture, with gestures often being used to do little more than emulate pointing and clicking on a desktop PC while more complex interactions are offloaded onto their 'Cortana' smart assistant system using speech recognition.

**Supporting Legacy Applications** An extreme case of the need for support of legacy interactions is the possible need to support the actual (unmodified) legacy application. Some projects, such as the MIT licensed uDesktopDuplication based on Microsoft Windows' Desktop Duplication API (see [30]), are intended to offer VC application developers access to traditional desktop applications on the host system. Microsoft's Windows Mixed Reality system also offers some support of legacy applications by design, generally using metaphors similar to the original ones such as a two-dimensional plane to project a window on and using pointing gestures in lieu of mouse clicks or taps on a touch screen. The support for the legacy applications themselves is rarely a focus for researchers. The need to accommodate legacy desktop software applications may be diminishing in any case: More and more of the most commonly used software products are now web-based

as-a-service solutions.
Nonetheless, there are use cases for which some measure of legacy support can be paramount. Business applications in particular are often decades old and extremely complex. If the industry were ever to embrace VC applications, support for such applications could be required since re-developing or reverse-engineering large parts of the legacy software is often out of the question. Even gradual migrations toward VC interactions would be eased if there was a way to integrate components that required legacy interactions with those which do not.
The main challenge for direct support for legacy applications within the VC is therefore the need for interaction techniques to be complete polymorphic equivalents: Whichever way the user interacts with the VC simulation, the goal would be to transform the input and output into what is required and provided by the legacy application. To this end, the most practical integration solution can often be the emulation of legacy interaction systems within the VC simulation, substituting equivalent VC-suitable polymorphisms of interaction scenarios for their legacy counterparts. Examples for this include:

- legacy windows being simulated as floating in mid-air or being rendered to a surface
- the use of transparency to allow for depth perception even through two-dimensional surfaces
- selection gestures using finger or gaze tracking systems rather than mouse pointers or touch-sensitive screens
- speech recognition instead of keyboard input

**Custom vs. licensed applications** One decision the developers of a VC application are likely to have to make often is that of whether to implement a local component of the system themselves or use an existing product. This section will elaborate on this. Note that non-local implementations, both custom and third-party, are discussed in a later section.
At least part of any VC application will always be based on third-party software used under a license. Contemporary software landscapes depend very heavily on the re-use of existing software solutions, especially as a foundation such as when abstracting away from hardware or creating specialized solutions based on a generic template. There are many types of licenses under which software may be used. Often there are also multiple licenses available for a single product.

Among the most popular licenses for research teams are open-source licenses, granting them the rights to examine and often also change and redistribute the code they use. Researchers should always be aware that these licenses vary very widely and should be read carefully before proceeding with the use of the product they pertain to. The term open source software (OSS) has different definitions and somewhat conflicting views exist as to what it entails. A comparison of different terms can be found in [24]. For the purposes of this paper, it is assumed that 'Open Source' refers to any software that is both 'open' and 'free' and conflicts with neither 'The Free Software Definition' by the Free Software Foundation (FSF) in [25] nor 'The Open Source Definition' by the Open Software Initiative (OSI) as published by them in [59].
Proprietary licenses vary at least as much in their terms as open-source licenses do.
Whether developing a custom solution is a sensible option depends on a number of factors. First and foremost, it would have to be an option in the first place: It may simply be impossible for the development team to develop their own equivalent due to a lack of time or personnel. This may be the most common reason for researchers to use proprietary software. Secondly, even if the time, effort and funds necessary for such a development were available, the re-use of software is often the most economical way to proceed.
Nonetheless, under some circumstances it may be prudent or even necessary to develop a custom solution. A proprietary one may not even exist or if it does, the opaque nature of its closed-source code might make it unreasonably difficult to adapt it to a specific use case, especially in research contexts - where the use cases are quite likely to differ from those envisioned by the creators of the software. There could also be privacy concerns preventing its use, or contractual terms which preclude its use in research contexts. A particularly common contractual obligation with potential for being disruptive is that of payments to the licensing party. If the price tag attached to the use of a proprietary solution is beyond what a research team can support, the only remaining option could be the development of a custom alternative[1].

### Requirements

The following challenges have been identified as being specifically relevant for the integration of local software into a VC application.

[1] Though attempts of re-negotiation with the licensing firm should often also be considered.

**Access Conditions** In order to integrate local software, be that standalone or in the shape of code modules, an important requirement that needs to be fulfilled is that of legal access to the software. Often, that means agreeing to contractual obligations. Researchers should bear in mind that closed-source software (especially standalone applications) may be difficult to integrate if it does not offer documented and ideally standardized interfaces for this purpose. In addition, even though it may be technically possible to wrap the UI of an application in such a way that its adaptation for the virtuality continuum does not require access to its code (see 'Supporting Legacy Applications' above), care should be taken to adhere to possible contractual limitations to this strategy. For example, contracts often forbid the 'programmatic' use of a software, making its integration in this manner illegal.
The simulation's own underlying architecture also contains potential objects of interaction, such as graphics rendering and file systems. This leads to the requirement that the simulation engine software actually makes these accessible in the first place. Simulation engines tend to be proprietary and almost completely closed-source. Since this is a very common requirement for developers (in the industry as much as in research), simulation engines generally expose a number of APIs to underlying systems such as audio and graphics rendering systems. Nonetheless, if this support is insufficient, it may be necessary to circumvent them and find other ways of interacting with the host system.[2] This may not always be possible - Web-based VR applications for example may not offer access to systems outside the sand-box they reside in for security reasons.

**Communication Requirements** Since the communication is limited to the local system or network, quality-of-service constraints due to the implementation of an integration are not a major concern most of the time. Instead, communication requirements tend to arise from the heterogeneous ways in which programs may communicate. Depending on the available avenues of communication, the simulation may need to communicate with other programs via the network, the host system's message buses or its file systems. In the case of 'non-invasive' integrations of third-party software, i.e.: without access to its source code, communicating with the application may even necessitate the simulation of user input (e.g. via touchscreen, keyboard or mouse) and processing of system output intended for the user (e.g. screen and audio capture).
The integration of local software modules (rather than standalone applications) is generally subject to fewer specific requirements, but due to their closer coupling with the

[2] More on this in section on Multimedia and Streaming, as well as the Omniscope project (see appendix).

simulation's code they may require even more extensive knowledge of their architecture, especially where APIs are concerned (e.g. the structure of the data they can produce and consume).

**Interaction requirements** Some local software may rely on interaction techniques specifically intended for traditional UI systems. Their integration may then necessitate a re-design of their user interfaces. To this end, new metaphors may need to be found, especially for entities with a purely virtual existence.
Intermediary forms of interaction designs based on translating between traditional and VC interaction techniques may prove useful here, especially if the integration of a legacy system has to be achieved without changing its own code (e.g. because it is a closed-source product under a proprietary license).

### 3.1.2 Inter-Networks

This category includes remote software systems such as internet resources and online services.
Services may be reachable via extended inter-networks, such as the internet. Some of them may be under the control of the development team, such as custom services developed in-house and deployed to the cloud, others may be services developed, operated and maintained by others (institutions, firms and individuals).

**Software-as-a-Service (SaaS)** concepts have become ubiquitous in recent years and deserve special mention in this instance, too. For an in-depth introduction into the concepts of Infrastructure/Platform/Software-as-a-Service infrastructures, refer to standard works such as [8]. This paper will focus on the use of Software-as-a-Service (SaaS) offerings.
SaaS offerings are typically software systems exposed to the internet. Their development, maintenance and hosting are performed by one or more companies with whom the user has to enter into a contractual relationship in order to be permitted to use the service.
Since they typically require little set-up and customization and are easily integrated with local simulation systems, they can be a means for extending a VC application efficiently in terms of programming effort. On the other hand, their (almost always proprietary) license and payment models can be problematic. Even more importantly, research efforts may require some amount of transparency - a requirement running counter to their

design, consisting mostly of entirely opaque and proprietary server-side code. Some companies use (web-)service approaches not only to make development and update efforts easier, but to safeguard intellectual property and know-how.
SaaS offerings are subject to many different pricing schemes: Some services are free, often maintained by non-profit and government institutions. Others do not require payment in the traditional sense but may use advertisements or commercially use and sell data generated by the use of its services. Some offer a part of their service for free, switching to a premium model based on the frequency or intensity of their use. Others may offer services for free to smaller companies, individuals or research institutions.
Since these contracts are so extremely diverse, it is essential for researchers to familiarize themselves with them before use. Failure to do so is often followed by a need to abandon a service which much of an application had been built around or even legal proceedings - which has been known to severely disrupt research efforts.
In addition to issues relating to the legal relationship to the owner of the service, another issue to consider is its stability: On one hand, the firms rendering the service may have resources at their disposal which the research team does not. That could be highly scale-able global infrastructures or specialized proprietary technologies. As such, they may be able to make their service offerings faster, more efficient and more stable than the research team could. On the other hand, since the running of the service is not under the control of the researchers, there are a number of situations that could cause unforeseen disruptions. The most basic of these is scheduled downtime, which - while often necessary for maintenance - might run counter to the research team's own schedules. Instabilities of the software itself, such as an inability to scale with higher load, could also cause the service to be disrupted. Some services have also been known to frequently be targeted by DDOS attacks for a variety of reasons. Perhaps the two most extreme disruptions possible are the discontinuation of development and maintenance and the complete cessation of the service, such as might be brought on by the bankruptcy of its owner.
Studies also suggest that many SaaS providers are subject to a great number of known security vulnerabilities. Some research efforts exist for unified security infrastructures with the aim of alleviating these problems (see [11]), but no such infrastructure has been widely implemented as of yet.
Some of the problems mentioned above can be addressed through appropriate Quality-of-Service agreements with the company rendering the service. Here, it is important to consider that a QoS agreement does not truly guarantee anything other than the liability of the service provider if the constraints are violated. Since a research team does not

generally follow a profit motive, however, contractual penalties (such as reimbursements) may not be of much help if the quality of the service falls below the agreed-upon levels. In addition, the possibility of a complete collapse of the company also need to be considered - in this case, not even contractual penalties would be applicable.
Given network access to the service and the acceptance (and fulfillment, e.g. via payment) of the legal contract, the first functional requirement for the integration of SaaS offerings is their integration with client software. Sometimes, client software modules/libraries for the networked service may also be available, though their own license conditions should be taken into account. Since the use of these services almost always requires a number of configurations to be set (such as login parameters, access tokens and proxies), a VC application will need to support efficient configuration, ideally with as little interaction required from the user as possible. The integration of configuration management tools could be useful for this.
Once connected to a simulation, the use of the service requires the development of appropriate interaction techniques and scenarios much like other software integrations. Due to the lack of control over the service, it may also be required to provide fallback scenarios in cases when the service is unreachable.

**Custom Hosted Services** As opposed to the SaaS offerings mentioned above, custom hosted solutions have been developed within the research context but are hosted by another party outside their local infrastructure. Many variants of hosting service exist, with varying levels of service and control, such as Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) offerings. The most popular way at this time is through containerized software instances in a computational cloud.
These services have the benefit of the software itself being largely under the control of the researchers; giving another institution (most likely a firm) control over the hosting can be beneficial as it can afford a level of stability which the researchers themselves might not be able to reach. For example, a solution hosted on the Google Cloud Platform uses Google's own extensive distributed computing resources and as such can be made all but immune to localized hardware and software failures.
The main drawback these services have over pure SaaS solutions provided by other institutions is the necessity of the researchers having to implement the service themselves, resulting in more time and probably funds being expended. Of course, in some cases, the additional control over the service's inner workings this affords them may actually be a benefit.

Other drawbacks of custom hosted services are mostly similar to the SaaS offerings mentioned above, especially where non-functional requirements are concerned (latency, jitter etc.). They do not result in quite the same amount of dependency on outside firms as the service's own implementation is under the control of the team, though being dependant on the hosting provider can potentially pose similar risks in terms of Quality-of-Service agreements and possible bankruptcies. Nonetheless, it is generally much easier to move one's software to another hosting service than it is to find an entirely new provider for a specific service.
It should be noted that the lack of control over the hosting infrastructure can come with a lack of control over security as well. Research suggests that - like SaaS offerings - IaaS and PaaS hosters are also subject to a large number of known vulnerabilities (see [11]). As in SaaS offerings, the integration of custom hosted services will require the client software to find the server, be configured to talk to it, make the correct requests and deal with the results. To this end, client software may be needed and configuration management can be useful. In addition, service discovery mechanisms may also work to alleviate some of the need for configuration by the user. This may require the use of an overlay network to connect (and confine) service discovery to relevant servers. In terms of interaction design, there are few differences when compared to locally hosted solutions.

**Web-based VC** Web-based VR applications are one way to better leverage the interconnectedness of computer systems. Two notable types exist: Browser-based VC applications and VC-based web browsers.
Browser-based VC applications can be hosted like any other web presence, being loaded on demand by navigating to them using a browser. An example for a way to facilitate this is the A-Frame framework ([1]), developed originally by Mozilla and now maintained by developers from Supermedium and Google. The framework allows for the development of VR applications which run directly in the user's browser, foregoing the installation of additional local simulation engines. To this end, it leverages the Three.JS API which in turn uses WebGL for rendering.
VC-based web browsers on the other hand represent something like the inverse of browser-based VC applications: A browser-like platform within a virtuality simulation which allows the user to browse the web much like a traditional browser while leveraging VC interaction scenarios. A prominent example at this time is Firefox Reality, a VC-based browser by Mozilla (see [52]).

### Requirements

**Access Conditions** Like local software, remote software can be subject to many licensing schemes; contractual obligations should be among the first requirements to be considered.
The ability to access the service's servers through the network is another vital access condition. This may be subject to regulation, e.g. by the research institute, the internet service provider or the state.
The decreased control over the software as opposed to hosting it locally can be cause for various concerns: There may be less control over when and how changes to the service might occur.
There are many documented cases of remote software services causing severe problems in conflict with requirements concerning privacy and security. As mentioned at the start of this chapter, VC systems have the potential of containing vast amounts of extremely sensitive and personal data - not only on the consumer market, but even and especially in research contexts. Granting a service provider access to this data then comes with requirements that this data be kept confidential. This requirement has to be reflected in the usage agreement. It should also be considered whether the possibility of the service provider violating this agreement (intentionally or unintentionally) is a significant possibility - for example if their service is subject to insufficient security precautions. The accidental leak of personal voice recordings by Amazon (see [5]) is an example for this.

**Communication Requirements** The integration of systems hosted outside a local environment tends to happen on a higher abstraction level, further from the hardware than the integration of local systems. This can alleviate some concerns, especially of those related to platform independence and access to the technological infrastructure of the simulation itself. There are challenges, however, which are more prevalent in internetworks. Non-functional requirements in particular often conflict with the Quality-of-Service offered by the services, their hosts and the network used to communicate with them. Requirements of particular importance are:

- Latency (e.g.: long round-trip times, delays)
- Bandwidth/throughput
- Reliability (e.g.: jitter, package losses)

- Security (e.g. authentication, data theft)
- Stability (e.g. unscheduled downtimes, discontinuations, breaking changes)

**Interaction Requirements** Much as for local software systems, the design of interaction techniques and scenarios which allow for integrated remote software to be used efficiently and effectively is a major requirement. In addition, it may be necessary to design these interaction systems to more easily accommodate and alleviate problems typical for remote system communications. Especially if the possibility of this integration not being available due to server downtime is to be expected, there may need to be contingencies.

## 3.2 Multimedia and Streaming

VC systems are multimedia applications. Auditory and visual stimuli in particular require streams of data between devices: At the very least, the host system normally needs to send rendered video feeds to its display and audio to headphones or speaker systems. In addition, gesture recognition, motion tracking, speech input and other systems stream their data in return, providing input for the simulation system. Integration of these streaming systems requires different technologies compared to most other software systems and therefore differs in its requirements. This makes it worthy of additional and separate discussion in this section.
Note that multimedia streaming is an extensive and complex topic going beyond the confines of this thesis. As such, the discussion will be confined to only those aspects which are of particular and noteworthy relevance for the integration into VC systems.
Besides streaming, another important aspect of multimedia in the VC is media analysis and transformation. Computer vision for example is needed to facilitate many core features such as motion tracking or feature/object detection.

### 3.2.1 Media Streaming

Multimedia streaming encompasses many standards, drivers and requirements. As such, streaming systems are rarely implemented completely from scratch. Inter-operability

of VC systems with other multimedia systems and peripherals should often be achievable using existing streaming frameworks, such as DirectShow (on Microsoft Windows), GStreamer (platform-independent, included in most Linux distributions), VLC (best known for the VLC player based upon it) and ffmpeg. However, these are not necessarily readily accessible to the developers of VC simulations: While IDEs such as Unity and Unreal often feature embedded media player variants capable of at least playing back media from streams, researchers may find these to be limited in terms of supported sources and formats and not extensible. For VC applications in need of extensive, customizable support for streaming media, it can instead be necessary to integrate local software (such as the frameworks mentioned above) for the task.
The Omniscope project (see [46]) can serve as an example: Since no solution existed that satisfied the exacting specifications for use in the CSTI, the Omniscope Multimedia Framework was developed to fill this gap. Omniscope is a multimedia streaming, analysis and transformation system intended to fit a variety of development workflows. Its primary use case is as a streaming media input module for simulations created with Unity. It is however also capable of working as a standalone executable, e.g. a media player with computer vision capabilities and easily adapted to other development environments. The Omniscope can serve as an example of how such a streaming infrastructure may be effectively integrated into a VC simulation.

### 3.2.2 Video analysis

The transformation and analysis of video data is a common requirement, especially in applications with graphical user interfaces. VC simulations are among the most challenging use cases, especially owing to the high quality requirements caused by needing to interface as seamlessly as possible with human perceptualization.
Computer vision in particular as an umbrella term for those technologies which permit computers to process visual inputs is needed for a number of use cases associated with VC applications. Outside-in motion tracking systems such as the ART and LeapMotion systems use computer vision algorithms to accurately measure objects' position and orientation in three-dimensional space. Inside-out spatial mapping sensors such as those of the Microsoft HoloLens use cameras and complex algorithms to create three-dimensional maps of the user's surroundings.

### Foveated streaming

"Humans see only a tiny region at the center of their visual field with the highest visual acuity, a behavior known as foveation. Visual acuity reduces drastically towards the visual periphery."[67].
Foveated imaging is a term encompassing techniques wherein quality (i.e. the resolution) varies across the image, depending on so-called 'fixation points'. These fixation points are areas of particular interest where resolution is of a higher importance than in the rest of the picture. Foveated imaging is very common, for example in digital maps: When a user moves the center of the map to a specific location, the first parts of the image that will appear (often individual rectangular images tiling the larger one) are typically those of the exact location, followed by more and more of the surroundings. As the image resolution is then incrementally refined, the center is always the first to be updated and improved. In short, foveated imaging can allow for more efficient use of network and processing resources by offering a way to prioritize parts of an image over others.
It is debatable whether completely disregarding part of an image can be considered part of foveating it. For example, this would include all the parts of the map in the previous example which were not shown but were part of the map. In this thesis, this will also be considered part of foveating it.
Foveated streaming in particular is a related concept not entirely unique to VC applications, but of particular importance to them. It refers to foveated imaging being applied not only to single images but a video stream. It includes the practice of tracking the user's gaze and adapting a video stream to it. This can be very useful for VC applications in a number of cases. The user's field of view is much larger when using a head-mounted display than a monitor would be. Head-mounted displays also spread their overall resolution over two eyes (generally two distinct displays), which required significant bandwidth and rendering resources. Foveated streaming can alleviate this need somewhat. A detailed example of how foveated streaming could be implemented can be found in [66].
Sensible placement of fixation points is one of the challenges of foveated streaming. Gaze tracking in particular is very useful for this and necessitates the integration of additional hardware and software. Another challenge is the manipulation of the stream itself. Depending on the streaming frameworks and pipelines used, it could be possible not only to display a foveated image but to negotiate a foveated stream from its source: The source would apply the foveation and only transmit an already foveated stream to the destination. This procedure could serve as a means to alleviate QoS concerns regarding the bandwidth needed for the stream itself. On the other hand, this can cause additional

requirements to arise as the user's gaze can shift very quickly and the re-negotiation of the stream's foveation would need to keep up with this. As such, it is less feasible to foveate at the stream's source depending on the user's gaze within their immediate field of vision. It may, however, be perfectly feasible to have the source cull areas that are far outside it. For example, showing the user a 360 degree video (a video stream rendered onto the inside of a sphere, with the user in the middle) is a common use case for VC applications. Since the user could not shift their gaze to an area of the stream located behind them without turning their head and possible their entire body, this may provide enough time for the simulation system to re-negotiate foveation from the source.

### Requirements

**Access Conditions** By and large, the access conditions of interaction objects which include or focus on streaming services are not much different from other local and remote systems: Some may arise from legal obligations, others from the technical accessibility of the services. The protection of data being streamed is another requirement. In addition, there may be other ethical concerns related to computer vision, such as problems recognizing people depending on demographic modalities (see [60], [63]).
Foveated streaming is a special case in the overlap between streaming and computer vision. Support for foveation can be essential for some simulations and at least provide a boost in streaming efficiency for many others. In order to allow for foveated streaming based on the user's gaze, gaze tracking systems are required as a prerequisite.

**Communication Requirements** Media streaming mostly forms a sub-set of internetwork based integration, as such the requirements mostly mirror those mentioned above. Some of them are of particular importance to streaming services: Reliability, for example, is often of a lower importance, as generally a lost package does not warrant the additional complexity and delay caused by a re-transmission. Throughput on the other hand is often extremely high, especially where the high resolution and very high refresh frequencies of modern displays in general and VC HMDs in particular are concerned.
Several open source multimedia frameworks exist which lend themselves to being integrated directly into the simulation engine, particularly as a part of an interaction design which requires working with streams. The Omniscope project exemplifies how such an

interaction can work. The communication between the simulation engine and the streaming framework can require significant effort, especially in order to process (produce or consume) their data interchange formats.
The integration of video analysis (computer vision) systems has different requirements depending on the analysis system is hosted. If it is hosted locally, such as on the simulation's host system or in its network, network constraints are of less importance than the capabilities of the available hardware. If done remotely, the network's quality of service may become a much larger challenge.
Since humans are quite sensitive to visual delays (as mentioned above, this has even been linked to the phenomenon of simulator sickness [74]) perceptible visual delays and jitter are often particularly disruptive. Communication requirements may therefore include the need for these to be kept low.

**Interaction Requirements** The interaction with streams themselves could mean having to create interaction techniques for controlling them, such as playback controls. Video analysis is used to make many interaction techniques possible in the first place, such as through camera-based tracking systems.

### 3.2.3 Audio Analysis

Aside from video, audio streams are also an important part of many applications. The development and use of speech-to-text (speech recognition) and text-to-speech (speech synthesis) systems in particular makes extensive use of audio streams. Spatial sound solutions, too, need to analyze and transform audio streams.

#### Speech Recognition

Speech recognition in the form of speech-to-text (STT) systems has allowed for companion systems such as Google's Siri, Amazon's Alexa and Microsoft's Cortana to be created and very quickly rise to prominence. There are a number of different approaches to STT systems, but regardless of the implementation: The synthesis of text from arbitrary speech in near real-time and with a high degree of accuracy takes significant processing power. Most such systems are not provided on-premise but on third-party hosting sites and the cloud. For the service providers, this has a number of benefits: It makes it easier to keep trade secrets, gives them more control over how their service is rendered and

gives them direct access to an enormous amount of user-generated data. Among other things, this user data can be used to further improve their service, especially to train the learning algorithms the service is based on.
Conversely, for researchers, using such third-party services may not be desired for a similar number of reasons, ranging from privacy concerns over performance issues (esp. the inevitable internet delay) to a lack of customizability and configurability. Hosting STT services on-premise - while quite probably more expensive - is possible and may actually have some synergy with VC applications: STT systems normally use the highly parallel nature of graphics cards processors to their advantage. VC applications are likely to need high-performance GPUs themselves, opening the possibility of sharing the same resource depending on the load. An example for an STT solution which may be hosted on-premise is Mozilla's implementation of the DeepSpeech architecture developed by Baidu and presented in [29].
STT is currently a rapidly evolving technology. Google Inc. has recently unveiled an STT feature for their Pixel phones which works locally, without the need for server-based STT services. A paper they unveiled a year prior to this ([43]) shows that they have made huge strides in achieving a high STT accuracy with a minimum of computational effort and space. While this may not be of much use to VR researchers at this point, being limited to Google devices, it shows that STT on-premise or even locally on the same device is achievable.
Many contemporary STT systems are based on machine learning algorithms. In order to understand speech, they need a lot of data to learn from, primarily annotated sound files of speech. The large corporations offering STT services - primarily Microsoft, Google, Facebook and Apple - all have ways of obtaining sufficient amounts of training data for the languages they are interested in offering support for. Open implementations, however, such as Mozilla's DeepSpeech implementation, may instead need to rely on data available in the public domain, which is more limited. Thus far, Mozilla's implementation has been used to train a model of the English language which provides a reasonable degree of recognition accuracy. Premade models for other languages are not yet available. While other languages with a large number of speakers are likely to follow before long, less prevalent languages (i.e.: those spoken by less people, or at least spoken by less people willing or able to provide annotated samples) are much less likely to be supported. It seems likely that this competitive advantage of the more common languages will result in a further increase to the rate of their disappearance. The Mozilla Foundation appears to have recognized this and has launched a project through which volunteers can provide

annotated speech samples in order to ensure support for their native languages.[3]

### Speech Synthesis

Speech Synthesis (Text-to-Speech, TTS) refers to the generation of synthesized speech audio from digital text. TTS systems are often combined with STT systems as a means for an application to reply to a user's input, such as with the aforementioned Alexa, Siri and Cortana systems. TTS does not directly involve the analysis of an audio signal. It also does not require as much processing power as STT. TTS can already easily be hosted and run on-premise. A popular open source implementation for this is the Festival Speech Synthesis system (see [10]).
The technical requirements and quality-of-service constraints for text-to-speech systems are comparatively fewer than those of STT systems. Challenges arise mainly if the required quality of the synthesized speech is particularly high - when it no longer just needs to be understood but feel natural to the user. This may be the case if TTS were to be used to allow the simulation to dynamically interact with a user verbally, such as when the user talks with a virtual, simulated person. Considering the VC development community's tendency to strive for more and more immersion, this could quickly become a relevant issue for VC applications.

### Requirements

The integration of speech-to-text services poses a number of challenges. This is further complicated by the different strategies for implementation - particularly local, on-premise solutions vs. cloud-based services - differing in their requirements.

**Access conditions** With the advent of Smart Assistants it has become increasingly clear that streaming services in particular can pose a significant security and privacy risk: The as-a-service nature of speech recognition systems means that audio information is sent, processed and stored on the service provider's or even third-party servers. There have been several well-documented cases of unauthorized people obtaining access to this data. Depending on the nature of the research and goals of a distributed interaction system, this may not be an acceptable risk. An analysis of how these risks pertain to their

[3] https://voice.mozilla.org

research is a requirement of the integration of these systems. This may also lead to the additional requirement of developing precautionary measures, such as the suppression of ambient sound.
Details of how third-party STT services are implemented are often opaque and not modifiable for the developers of the VC system, though whether that is a problem would depend on the individual use case.
Self-hosted systems on the other hand come with less privacy and legal issues. They do, however, add a number of different requirements. In particular, running a speech-to-text service typically requires a trained model for the target language. If no sufficient model is available, it would need to be created. Creating such a model requires extremely large amounts of annotated training data sets and significant processing resources.

**Communication requirements** Cloud-based as-a-service solutions for STT are typically fast and efficient to integrate, easy to maintain and likely to have a greater degree of accuracy than self-hosted alternatives. On the downside, their challenges can include various QoS requirements due to their streaming nature (similar to video streaming requirements, see above).
Each STT service uses different exchange formats and APIs. Some do not even use audio streams in the stricter sense but only offer analysis of a sound file. Adapting to the service's API can be a significant effort, especially if there is a need to record and transform audio data to their specifications.
For cloud-based services, QoS requirements can apply to STT systems in much the same way as for any other remote streaming service. Self-hosted services run in a more controlled, local networking environment where quality-of-service constraints are easier to manage.
The integration of TTS systems is similar, though somewhat easier in many cases. For example, while an STT solution often provides a number of possible interpretations of an analyzed utterance, TTS solutions only create a single sound stream or file synthesized from a single text.

**Interaction Requirements** Both speech-to-text and text-to-speech services add a range of new possibilities for interaction scenarios. Their use requires the selection or development of interaction techniques. For example, the use of a speech-to-text service to analyze data gathered using a microphone in the user's HMD could result in a set of textual interpretations and corresponding confidence values. Interactions using STT

require algorithms for choosing the most likely candidate thereof given the circumstances, assigning an intent to the text and in turn triggering an appropriate reaction to it.
As part of this (and any other) interaction's feedback channels, text-to-speech systems can be used to convey a message to the user. If the interaction scenario is based on letting a human interact with a virtual human (or anthropomorphic personification), using both TTS and STT in combination can become a requirement.

## 3.3 Cyber Physical Periphery

Cyber-physical systems (CPS) are defined in a variety of ways. In probably their most basic definition, CPS incorporate physical and software components. Many definitions also include human factors, especially that of a user of such a system. They also generally agree on the physical nature of system components having a significance outside the need for hardware which the software may run on.
Outside of that, however, the definitions start varying extremely widely. Furthermore, they often slide into hopeful visions of what CPS systems may be used for and end up looking like. For example, the US National Institute for Standards and Technology defines CPS as: "Cyber-Physical Systems (CPS) comprise interacting digital, analog, physical, and human components engineered for function through integrated physics and logic. These systems will provide the foundation of our critical infrastructure, form the basis of emerging and future smart services, and improve our quality of life in many areas."[81]
Taking into account this lack of taxonomic clarity, the following definitions are going to provide the basis for this thesis' discussions:
Cyber-physical peripherals for the purposes of this thesis are devices with hardware and software components in the user's physical surroundings which the user is able to interact with physically as well as virtually (through inter-connected computer systems). As opposed to the aforementioned virtual periphery, they therefore also contain hardware elements which the user may interact with directly (i.e.: physically). They are considered to have the ability to provide the simulation system with information about their status, e.g.: Temperature, position, rotation or light intensity. Furthermore, they may function as actors, influencing the user and their environment (e.g. heating the room, opening shutters) or providing perceptible input to the user directly (e.g. vibrating in their hand).
Cyber-physical systems are related to the terms Internet of Things (IoT) and 'smart'

something (e.g. smart device, smart home, smart environment, smart city). Definitions of these terms often intersect and change over time. The IoT in particular is a term that has changed in meaning many times for the better part of two decades (see [23]).
In this thesis, the term Internet of Things or IoT is going to be used to refer a network of cyber-physical devices interconnected with one another as well as other software systems and users.[16]

### 3.3.1 VC interaction devices

VC interaction devices are those cyber-physical devices intended and designed specifically to allow users to receive stimuli from the simulation system or provide input for it. This includes head-mounted displays (HMDs) as well as wands or tactile feedback wearables. VC interaction devices are cyber-physical devices by definition: Since the user uses them to interact physically with a non-physical system, they act as an interface between the physical and the virtual.
It should be noted that a VC interaction device does not need to be interactive itself for it to enable interaction with the simulation. Cameras, for example, may serve only as a recipient sensor of visible user action while not offering any feedback channels themselves. The user does not interact with the camera, they merely interact with the system by means of the camera (as well as other devices). Interaction using cameras is then an emergent function of the VC system as a whole which includes other (cyber-physical) devices for the generation of stimuli. Nonetheless the camera could be considered a cyber-physical device due to its physical nature (e.g. position, rotation in real space) being an important factor of its utility.
In some books such as [17] these devices are categorized based on whether they are intended to provide the user with stimuli or receive input for the system from them. However, these devices are typically multi-modal as well as bidirectional: They do not only provide system input or stimuli, but both. This thesis will not categorize them further based on sensory modalities but instead focus on their cyber-physical nature and the requirements of their integration and use.
Most cyber-physical devices in general may be used to interact with the VC system regardless of their original intended purpose. VC interaction devices in particular only differ in that this interaction is their primary (possibly their only) purpose.
Contemporary VC interaction devices - while cyber-physical in nature - can rarely be considered part of an Internet of Things. This is due to their purpose being closely linked

to the simulation they are intended to enable interactions for - this does not generally require anything beyond a close, direct interconnection.
The head-mounted display which is central to most immersive VR and AR applications is a prime example for a cyber-physical VC interaction device. It encompasses sensing components (such as gyroscopic sensors for rotation) and acting components (such as stereoscopic screens). The user interacts with it and it is connected with the VC simulation. Its sensors allow the simulation to track the position and orientation of the user's head. Its stereoscopic displays are used as an output channel for the simulation.
In research contexts, the creation of VC interaction device prototypes and experiments is not uncommon. Examples include the VibeGlove (see appendix) or the head-mounted thermal feedback device described in [62]. An approach for providing tactile feedback to users using VC interaction devices has been suggested by [37]: The use of miniature drones. These drones' positions could be compared with the user's motions using tracking markers. To simulate the sensation of objects colliding with the user, the drones could be steered to precise positions within the tracked space.

**Example: VibeGlove Experiment** The VibeGlove is an experimental tactile feedback system in the form of a glove. It was intended to showcase an efficient way (in terms of component costs as well as development time/effort) for an accurate tactile feedback system to be implemented as part of a research project.
The glove is comprised of five small, coin-shaped vibration motors (originally intended for use in phones), a minimalist driver circuit and a microcontroller. The vibration cells and circuits could be affixed to the user's fingers and wrist by means of velcro straps. It was programmed using Arduino in order to stay beginner-friendly for novice-level developers and programming laymen. Communication between the glove and the simulation system was achieved via USB, using a native serial transmission library tied into the Unity-based simulation as a native plugin.
The user's hand system were tracked using the LeapMotion motion capture system. The VibeGlove's minimalist, skeletal design did not interfere with the LeapMotion system (at least not to a noticable extent).
The simulation's entity component system was extended by tactile feedback components: Attaching collision detection algorithms to entities visible in world space as well as the virtual representation of the user's fingers made it possible to detect whenever a vibration stimulus would be necessary. The vibe-glove object was then notified using a direct serial connection (Mk1) or the MQTT messaging system (Mk2) and made to activate

the vibration cell on the corresponding finger.
The integration of the VibeGlove system showcased two different avenues of communication: A serial (USB) connection wired directly to the simulation's host system and a more loosely coupled publish/subscribe system using MQTT. Initial testing did not reveal a noticeable difference in trigger delay between the two. It did, however, show that the loosely coupled Mk2 system was more easily extensible with additional systems due to the subscription to an event being disconnected from the VC simulation. The simulation itself only had to publish events whenever a collision occurred without knowing or instructing a specific recipient system.

### 3.3.2 Cyber-physical device networks, smart environments and the IoT

Smart environments such as smart homes are often used as examples for an IoT. To a large part, they consist of cyber-physical systems.
In their technical implementation, these networks vary widely: They may use the ubiquitous IP-based networks (e.g. WLAN), specialized variants thereof (e.g. 6LoWPAN) or something else entirely (e.g. SIGFOX, Semtech's LoRa) in any number of topological configurations. Correspondingly, all attributes of their networks also vary widely.
There is a number of commercial smart environment and smart home systems. These are often subject to vendor lock-ins, not standardized and/or incompatible with other vendors, leading to a strong degree of technological fragmentation. These systems generally use a server as a hub for their interconnection: A local 'smart home controller' or a remote server. Some devices however (especially those by smaller vendor companies) use communication strategies based on open technologies and standards such as Bluetooth, REST-based web services and MQTT- A few also don't require a central hub - particularly those connected directly to the internet (e.g. some of the current Xiaomi/Aqara line of smart light bulbs).
Similar to proprietary smart home controllers, a widely recognized and popular example is OpenHAB, an extensive (and open source) smart environment server system. Its open-source nature and third-party binding system have allowed its' user and developer community to make it compatible with an extensive range of domestic cyber-physical devices.

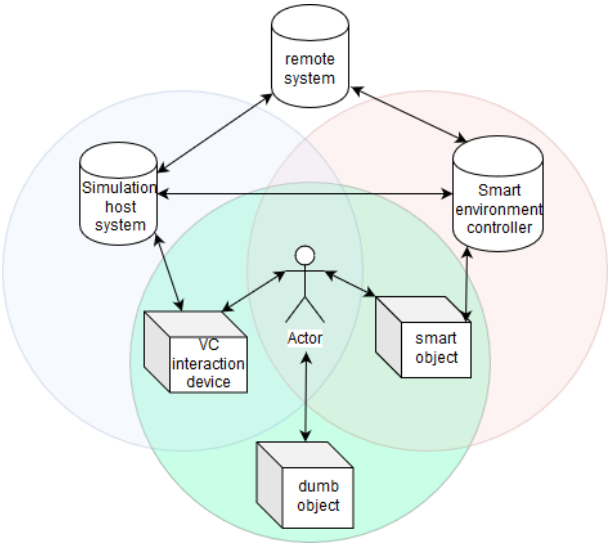



Fig. 3.1: Parallels between the VC's interaction devices and cyber-physical/smart devices. Green: Physical Environment, Blue: VC system, Red: IoT/CPS

**On the parallels between Cyber-Physical/IoT networks and the VC**

As mentioned before, VC interaction devices are cyber-physical devices, but are not generally advertised as such. A simulation connected with VC interaction devices is similar to a cyber-physical/IoT network: Both consist of networked, distributed computer systems of which at least some are integrated in physical object the user interacts with. Fig. 3.1 shows a simplified model of this symmetry.
This similarity leads to the conclusion that a unification of the two fields could lead to a more efficient integration of cyber-physical objects in general and VC interaction devices in particular: If a level of abstraction existed over the functionality of cyber-physical devices, it would make little difference which purpose a device was originally built for. Instead, its services as a sensor and actor could be used by the simulation or other members of a cyber-physical network on demand. A wand intended for enabling gestures in a VC system for instance may then also be available as a component of a smart home system and be used as a TV remote for a smart TV. A smart speaker (e.g. Amazon Alexa) on the other hand may serve as an access point for verbal interaction inside a VC application. Polymorphic interaction scenarios could lead to a solution for how this abstract offering of services can be defined efficiently.

**The Shelldon Project** The Shelldon Project highlights some of the most important aspects of a cyber-physical system. On a software and network communication level, the Shelldon prototypes can communicate with their shared host station (in a hub-and-spoke architecture). Using this, they can exchange messages with one another. On a physical level, they can also communicate directly with their users using visible light emitted from their LED matrices. They also receive physical input through rotational (gyroscopic) sensors and use this input to calculate the deviation from a 'resting' position (i.e.: when they are lying flat on their belly).
Integrating the Shelldon prototypes into a VC application would have to begin with setting up communication with their host station. This host station acts as an ad-hoc wireless LAN access point. Connecting it to the simulation would therefore either require connecting the simulation's host system to this wireless network or connecting the Shelldon's host station to another network besides its own. This could most easily be done via a LAN cable, though adding another wireless modem to it would not present much of a challenge either as USB WLAN modems are readily available.
Once this connection were established, the first interactions between the simulation's users and the Shelldon prototypes could be built. However, in order to make use of the Shelldons' cyber-physical nature, it may also be necessary for the simulation to be aware of their position and rotation. The Shelldon's own sensors cannot provide this information: They do not know their location or their heading.
As such, it may also be necessary for the simulation itself to track the prototypes. This could be done using marker-based tracking (see [45]) or, with significantly higher effort, markerless camera-based object recognition.

### Requirements

**Access Conditions** Much like any other system mentioned above, cyber-physical systems can be subject to license agreements prohibitive for research contexts. CPS devices can also have the built-in ability to gather data on their surroundings (using their own hardware and software), making them a potential threat to data security and privacy.
CPS often rely on server-side infrastructure hosted by the vendor (or even a third party). In this case, the same ethical and privacy concerns apply to them as for the integration of any other remotely hosted software system.
Depending on the use case, it may be necessary to gain access to a CPS device's internal hardware or the software powering it it its servers. Whether any of this is available (and

legal) varies between vendors.
Cyber-physical systems may also have additional information (data) associated with them on a virtual level which they themselves cannot offer: The Shelldon prototypes, for example, cannot themselves provide a simulation with a description of how they look. Since they are based on a three-dimensional model, however, this model can serve as a template for how the Shelldon prototypes are shown inside a VC simulation. Access to such additional information and documentation can be a requirement.

**Communication Requirements** The extensive platform fragmentation and lack of standards can make cyber-physical/IoT devices difficult to integrate. This issue is not exclusive to their integration with VC systems. Since this is a well-known and documented problem, standards are being developed and proposed on a regular basis. At best, this may lead to a solution at some point - at worst, however, it may instead further increase platform fragmentation. Regardless of the eventual outcome, the need to accommodate different interfaces (APIs) for nearly every vendor (if not every device) is a major communication requirement.
In addition, quality-of-service requirements can be an issue for many devices as well. This is most notable for devices intended to be used for user interactions which are particularly intolerant of latencies, such as head-mounted displays. Non-functional requirements such as bandwidth and latency can be particularly limiting for cyber-physical systems as they are subject to severe technological trade-offs, particularly between power efficiency, transmission range, bandwidth as well as computational power. A cyber-physical device communicating using a LoRa transceiver module, for instance, might have a range of upwards of three kilometers but may be limited to a few kilobytes of data per seconds. On the other hand, a device connected to a wireless LAN access point can send and receive data at a speed several orders of magnitude faster while being limited to a range of a few dozen meters.

**Interaction Requirements** The wide range of devices and purposes encompassed by cyber-physical systems makes their interaction requirements equally diverse. The integration of a CPS can require or allow for a range of new interaction techniques.
The most important interaction requirements are those pertaining to allowing the user to effectively interact with the physical nature of the objects in conjunction with the virtual. In some cases, the integration of a cyber-physical device may also require the integration

and use of additional systems. An example for this is the use of camera-based tracking systems to gain additional information about its position and rotation (see below).
Depending on the use case, there may also be non-functional requirements attached to interactions with a cyber-physical system. It might, for example, be required that there be no mismatch between the perceived location and orientation of a real object and its virtual counterpart.

## 3.4 Supporting unknown spatial environments and dumb objects

Not every part of the environment the user interacts with in a VC system is 'smart'. The floor they stand on, the table they sit at, even the cable they trip over are unlikely to be able to inform a VC system about their status automatically. Yet their integration into the simulation system can be sensible if the user wanted to use them - or avoid tripping over them.
A 'dumb' object is a juxtaposition to a smart object in that it does not have any inherent ability to be interconnected with the simulation, offer information about themselves or be called upon as an actor. This can be misleading, since the act of accounting for it often means that the simulation will create a virtual representation of the object - according to some definitions, this could cause them to be considered at least cyber-physical objects themselves. In this case, we will not conflate the two in this manner. This section deals with the integration of objects which are incapable of providing the simulation with information on their own. That can also include cyber-physical devices whose capabilities for integration are too limited for a use case: The solution can be an augmentation of their abilities using separate systems.

### 3.4.1 Spatial mapping

Spatial mapping encompasses those technologies which can be used to map the physical shape and topology of an area. For example, the Microsoft HoloLens augmented reality headset is capable of using a number of infrared cameras to form a three-dimensional representation of the user's environment in real-time.
Spatial mapping is a highly complex process, typically relying on computer vision algorithms to extrapolate distances between different camera feeds in much the same way as

creatures with multiple eyes do. This is often particularly difficult in areas with a lot of reflective surfaces, though this can sometimes be mitigated by using cameras sensitive to specific wavelengths.
The result of spatial mapping is generally a mesh connecting individual extrapolated points in space. This may be done in advance, for example if the simulation is intended to display an immersive digital representation of a real place. In other cases, the spatial mapping is done at (or near) real-time. Here, the focus is on gaining a digital representation of the user's immediate physical surroundings. This can be used to then selectively facilitate apparent interactions between virtual entites and the user's real physical environment. For example, the user could throw a virtual ball which could then rebound off the spatial mesh.
Spatial mapping systems are sometimes capable of rendering textures onto the generated mesh, providing the basis for an even more immersive virtual representation of the user's physical surroundings. The current HTC Vive Pro HMD (with its corresponding software packages) for example is capable of using its stereoscopic cameras to create a fully textured spatial mesh.
The real-time use of spatial mapping is probably the most important aspect of this technology where distributed interaction scenarios are concerned. It enables the on-the-fly integration of a previously unknown spatial topology. Integrating this further with other known facts about the user's environment, such as floor and building plans and CAD-files, can enable a variety of interaction scenarios. For example, a construction worker wearing a head-mounted augmented reality display with real-time spatial mapping could see the plans of a building (such as a wall with an embedded gas pipe) overlaid over its physical state while it is under construction. This would enable them to drill holes while being able to avoid hitting the embedded gas pipes.
Another challenge for spatial mapping technologies is the recognition of an area, re-using already known spatial topology. For example, if a user were to return to a room that they had already been in and which was already mapped, it would make sense to re-use this mesh and only make adjustments where needed (e.g. because objects had been moved in the meantime).

### 3.4.2 Object recognition and registration

Often, markers are used to aid in the recognition of objects. For example, a two-dimensional image applied to an object's surface in conjunction with multiple cameras

whose position and rotation are known can be used to extrapolate the position and rotation of the object. For an introduction into the different types of tracking systems, see [45].

### 3.4.3 Manual design

While it may be possible to leverage sensor systems such as spatial mapping to glean information about them, sometimes the virtual environment needs to be handmade to fit congruently onto reality. As mentioned above, CAD models and floor plans are examples of other virtual representations of real objects. The SpaceFlight experiment gives examples of how virtual representations of real objects could be manually created and positioned as part of the simulation's own creation rather than at runtime (see appendix, [6]). In [33], Jo et al. propose a planning toolkit to simultaneously design virtual and real environments to match. While not as flexible an approach as real-time mapping of the environment during the runtime of the application, models and plans can sometimes be more useful. For instance, they are often much more detailed and accurate than the results of on-demand spatial mapping.

#### Requirements

**Access Conditions** Aside from potentially needing to obtain (especially buy) them, gaining access to the objects themselves is rarely an issue for the integration of unconnected physical objects into the distributed interaction system. Documentation and associated data, however, such as CAD models, can be more difficult to obtain.
The way in which physical objects are integrated might be able to open the system to ethical and privacy concerns. Particularly camera-based tracking systems (such as those in the Vuforia framework, [84]) can be problematic due to privacy concerns: They generally capture far more than just the object they were intended to track. This could, for example, include the user as well as any and all parts of the research laboratory. If this data they generate is made available to third-party service providers, care should be taken that this does not go against the laboratory's privacy and data protection obligations.

**Communication Requirements** Since direct bi-directional communication with these interaction objects is impossible on a software level, other communication is required. This is generally achieved through the integration of different systems, with their own requirements. In particular, their integration often requires that the system have prior knowledge of the object and recognize it using sensors. As a result, streaming services would need to be integrated as well, which leads to an inheritance of these streaming systems' requirements. If the system is intended not just to recognize and show an object but allow interaction with it, it may also be necessary to integrate actors as well as sensors. For example, a VC simulation may include the use of robotic actors to manipulate a real object. This requires the integration of these actors with their own communication requirements.
Much as for cyber-physical objects, users will need to perceive unconnected objects in such a way that their physical interaction with the objects is not negatively impacted. Since the user interacts with their environment and dumb objects in a physical way while being shown a virtual representation of them, the speed and quality in which this virtual representation is rendered can be an example for a limiting factor. A juggler, for instance, who tries to juggle real balls while watching their virtual representations lag behind would have a hard time timing their movements correctly.

**Interaction Requirements** The interaction with unconnected objects and their physical environment is mostly dictated by their nature, not by the simulation. The interaction design of their integration is therefore primarily concerned with enabling this interaction. For example, if a user wearing a head-mounted display were to step close to a real table, the interaction design of the integration of this table could consist of the simulation piping a stream from the HMD's frontal cameras through to the screens, allowing the user to take appropriate actions (i.e. avoid collision or pick up an item on the table). Some interaction scenarios however use the recognition of a real object as a basis for enhancing the object in a variety of ways or only using it as part of a more complex interaction. A tracked two-dimensional image captured through the HMDs cameras could be used to display a separate virtual object above it, with the user's interactions then focusing exclusively on the virtual representation.

# 4 Task-oriented development process

The interdisciplinary, often novel and unique challenges to building VC systems mentioned throughout this thesis culminate in a complex development process. While it is the primary goal of this thesis to shed light on the technical challenges of integrating systems into the VC in order to facilitate new forms of interaction, a brief discussion of some of the peculiarities of the development process of VC systems may serve to highlight some of the most important procedural challenges.
The focus of this chapter is the discussion of a task-oriented (or goal-oriented) development process for VC applications, particularly of how it differs from the way in which VC aplications are usually developed. This task-orientation is refined and incorporated into a proposal for economical VC applications.
Some of the interdisciplinary aspects to VC systems development are immediately apparent: Aside from software development in general, cognitive sciences, psychology and biology are fields of research that overlap at this intersection. There are also important interfaces toward sociological, ethical and political issues as well (see ch. 3). Another field of paramount importance is that of game design and development, whose proximity and overlap warrants it being given particular attention.

## 4.1 Games and their development

Most of the VC systems currently available originate in the entertainment industry, particularly in computer gaming and pornography[1]. The goal for computer games development could be summed up as the efficient entertainment of people for profit using interactive computer technologies. Since an introduction into how games are developed at all is beyond the boundaries of this thesis, refer to standard works such as [69].
This section aims to highlight some of the aspects of VC system development in the

[1]The pornography industry is not in the focus of this thesis, but in most central points where the VC is concerned it is not too different from the games industry in any case.

games industry which differ from software engineering in general. In order to do this, it needs to give a brief introduction into the sales schemes of the games industry and how they relate to how whether their products are developed to be throwaway products or maintained over decades. It also endeavors to develop an idea of whether distributed interaction systems may have something to offer for games and which challenges may need to be overcome that are unique to this field.

### 4.1.1 Sales schemes and their impact on sustainable development

In the computer games industry, the focus during software development seems often to be not so much the long-term running and maintenance of a single software, but the creation and sale of different individual products. This has a (probably somewhat circular) relationship with the traditional pricing schemes of this industry: Most often, many copies of a game were sold to individual customers at relatively low prices. The comparatively low time it took to play through the entirety of the game made it more profitable to release entirely new games rather than expending resources on improving a game already on the market and further along its (generally short) lifespan.
In recent years, though, the industry has diversified its pricing strategy somewhat. One example of this is the offer of purchasable extensions to a base game known as Downloadable Content Packs (DLCs), a pricing model exemplified by Publishers like Paradox Interactive. Here, the base game is typically enhanced significantly, for example through an additional available storyline, campaign or mode of play. This tends to increase the users' willingness to stay loyal to a game and/or play it again once they have finished it (a.k.a.: replayability). Another strategy is that of premium models which in some cases may not even require an initial purchase (then known as Free-to-Play and often colloquially referred to as Freemium) but try to entice the user into buying additional content in-game. This most commonly takes the form of many small 'microtransactions' which typically unlock only very little content, such as clothing and equipment for a player's avatar. Another form of this is the concept of teasing the player by letting them obtain or find 'locked' items in the game whose contents are unknown to them but hinted at being particularly valuable. Getting those contents then requires the user to buy a 'key' to 'open' it. This is commonly referred to as a 'lootboxes' scheme and considered to be a form of gambling in some jurisdictions, which have issued advisory notices or even placed legal restrictions on the use of this strategy (see the statement of the Belgian Ministry of Justice in [26]). Note, therefore, that gambits for the 'loyalty' of players that

come with these premium models are often considered to be ethically questionable as they often exploit psychological affects such as addiction and sunk-cost biases. This is an important avenue of research, but obviously beyond this paper's purview. It does, however, serve to illustrate that the way in which simulations (esp. games) are made to engage users should be viewed critically from an ethical perspective - even in research contexts, as their researchers' work setting the cornerstones of this technology gives them at least partial responsibility for the type of simulation users end up being exposed to.
Yet another pricing scheme which seems to be on the rise is that of a platform subscription. The user pays a monthly fee to a platform provider - typically this is the monopolistic vendor of much or all of their system landscape, e.g. Sony (Playstation platform) or HTC (Viveport platform). This vendor then provides them with access to any and all software products (games) in a large collection. The platform provider in turn redistributes a share of their earnings among the games' individual publishers and developers. This resembles the successful emerging streaming media publishing strategies of this time, such as Netflix, Amazon Prime and Spotify.
Successes and failures of many game companies over the last two decades indicate that long-term sales schemes aimed at keeping users loyal to a product for as long as possible (rather than bringing out additional products) have demonstrated a more effective return on investment and a more stable avenue for business development.
Therefore it seems likely that even though the games industry still does not seem to value maintenance and sustainability of their products as highly as many firms developing business IT systems do, this change in sales schemes indicates a rise in more sustainable development practices. This is likely to benefit VC researchers relying heavily on the tools and products of this industry in their own research and development.

### 4.1.2 Games and Distributed Interaction Systems

Distributed Interaction Systems would doubtlessly offer new possibilities for games development, too, but it seems that much of their potential is unlikely to be leveraged any time soon. This is mainly due to the development landscape being fragmented between those firms providing technological bases for games development. Firms like Google, Microsoft, Samsung, Valve and Unity can be seen as examples of very slim market supply diversity as they dominate their respective markets. As each tries to cement their hold on technological markets and maintaining a vendor lock-in (see [7]) on their products and platforms, innovative entries into their market have a hard stand and often end up

being bought up by them or never achieving enough market share for their continued existence. Virtual reality hardware in particular can be seen as an example for this, as Google, Microsoft, Samsung and Valve vied for dominance with their own offerings rather than focusing on cooperative solutions. Another particularly extreme example is Sony's Playstation console. From the VR headset over the computer's hardware, its operating system, development frameworks and software store to purchase games from: Everything belongs to the firm. Third-party vendors have next to no chance to place a competing product here. In addition, firms developing games for consoles like the Playstation are subject to exceptionally broad non-disclosure agreements making it all but impossible for developers to discuss any implementation details of their games.
There may be a gradual reversal to this trend where interoperability of VC hardware on the major operating systems is concerned. Limited cooperation between the larger VC hardware vendors exists now in the form of standardized APIs (see ch. 5), to be implemented primarily by hardware vendors to give software developers a uniform interface to develop against. A possible explanation of this is that the emerging market of VC technologies is so small and the initial investment required is so prohibitively high (especially for private customers) that only the combined software market for all hardware platforms (e.g. HMDs) was considered to be sufficient for enough customers to buy their products in the first place. These open standards are elaborated upon in ch. 5.
Games developers today cannot easily implement the interactions driving their games in a way that would be abstract, generic and standardized enough to allow for users to utilize their own distinct and possibly unique set of interaction objects. For instance, a user can not initiate a selection gesture in a game's menu using whichever smart assistant they possess (e.g. by verbally referring to the menu item) or substitute one manufacturer's interaction wand with that of another. Similarly, researchers face challenges integrating experiments into this landscape: It is exceedingly difficult for a researcher to replace a game's input device of choice with a custom variant in order to perform research on possible improvements in usability.
It therefore stands to reason that much like researchers of the virtuality continuum, the games industry is subject to similar challenges and stands to gain from possible solutions for the integration of distributed interaction systems.

The simulations developed in research contexts differ in a number of ways from those in the games industry, but generally they use the same tooling and basic technologies, such as IDEs and engines. They, too, often develop products with a relatively short lifespan:

Prototypes and experiments are often not needed after a study has run its course.
One major difference is that the systems developed in research contexts may be based on an unusual system landscape. In particular, they may require the integration of devices and systems not available to consumers. The games industry's tooling can cause problems in these cases, as it is mostly intended to offer support for common technologies.
Another difference is in the goal of the system itself: In research contexts, the VC simulation may be intended to be entertaining as well (e.g. in order to measure the effectiveness of interaction scenarios for games), but many other goals are possible, depending on what is to be researched. Immersion and presence in particular are considered paramount for VC-based games, but their pursuit can be inefficient for research contexts.

## 4.2 Task-orientation

Contemporary VC applications are centered around entertainment (rather than productivity), with most of the market divided up between computer games and pornography. On the scientific front, the primary research focus is to make the simulation more convincing and further blur the lines of what the users perceive as real.
There is some research being done on the use of VC applications for other purposes (e.g. their use in workplaces, businesses and industries), but it could be argued that the ubiquitous pursuit of improved realism and immersion is detracting from the research into using VR for other goals. For example, the goal of the simulation, often a task to be performed by the user, may not actually require high immersion or presence[2]. In addressing development inefficiencies and promoting task- or goal-orientation, immersion may have to take the back seat. This trade-off is related to the aforementioned realism trade-offs as proposed by Jacob et al. in [32]. Researching this trade-off further and finding ways to create simulations which focus on achieving well-defined tasks efficiently in terms of development resources is an important avenue of scientific effort.
The concept of task-orientation has already become a guiding principle for the CSTI laboratory.

[2]The term 'task-orientation' is therefore also used synonymously.

## 4.3 Economical VC Applications

This thesis proposes the following definition for economical VC applications as a basis for efficient task- or goal-oriented development.

### 4.3.1 Characteristics of an economical VC application

The following attributes are suggested as being characteristic of economical VC applications.

1. Goal orientation: Defined, clear goal(s)[3] to be achieved through the operation of the application
2. Minimalism: Foregoing the development of any features aimed at realism and immersion if reaching the application's goal is not significantly aided by them.
3. Economical development: Careful, efficient and prudent use of development resources

All three attributes define the limiting factors which are meant to keep the development of a VC application from becoming inefficient. They answer the following questions respectively:

1. What are the goals to be achieved?
2. Which features are required of the application to achieve them?
3. Which development resources are required for the implementation/provision of these features?

[3] Often but not necessarily institutional or business goals

# 5 Toward integration

This chapter deals with a number of strategies and technologies which could be used to to aid in the integration of different interaction objects. The list of these is not exhaustive and not intended to provide a one-size-fits-all solution, but as a proving ground for the requirements catalogue established in chapter 3.
Each strategy represents a different type of approach for integrating a VC system with elements of a larger system landscape, with the goal of enabling and aiding user interaction with new objects of interaction. Its potential for addressing the requirements introduced in chapter 3 is discussed and existing implementations (and relevant technical foundations) are referenced wherever applicable.
The following categories of integration solutions will be discussed:

- Open Standards and APIs for the VC
- Highly integrated IDEs
- Inter-systems communication and integration platforms

## 5.1 Open Standards and APIs for the VC

Ideally, the groundwork of solutions for interconnectivity and compatibility between systems in the VC would be set by universally agreed-upon standards. Interaction devices, for example, could make their sensor and actor capabilities accessible through a standardized API.
This section will use the term 'standard' to describe a way of integration and communication "established by authority, custom, or general consent as a model or example" ([50]). In a narrowing of this definition, it only refers to those solutions actually *available* to everyone, i.e. open standards: those which are openly accessible and legally usable by all developers and researchers of VC applications.

An example for an integration framework based on standards can be found in [71]. In their paper, Shrestha et al. propose a standardized framework for the integration of domain-specific application into the IoT. It is based on the Quantum Lifecycle Management framework as described in [23]. Similar standards could be developed for the integration of interaction objects into a distributed interaction system - in fact, the utility of smart IoT-devices in the domain of VC applications may even make QLM in general and their proposed integration in particular worth exploring as an option.
There are numerous benefits to standard-driven systems integration. The communication effort between the many hardware and software developers and vendors could be significantly lessened. Without openly accessible interfaces, the need to register with vendors, get access to their documentation and then implement the necessary routines to interface with their hardware can be a significant hurdle.
Unfortunately, universal standards (especially concerning access to their systems) are often considered problematic by businesses. Some, like Microsoft, are even known for actively undermining standards, such as with Microsoft's 'embrace, extend, extinguish' doctrine (see the findings of the US Department of Justice [80]).
As mentioned previously ('The Development Process', section on games development), the dominance of relatively few large firms in the realm of VS systems in general and hardware devices in particular also seems related to this situation. A root cause is the problem that these firms can communicate with each other directly relatively easily while locking out anyone else not able to do so with opaque and proprietary interfaces as well as extensive non-disclosure agreements. New competitors have a hard time entering and existing in the market. The established market leaders' control over market access, market shares and - subsequently - business value can be used to force the newcomers to cooperate - or be purchased outright.
The establishment of open standards and frameworks can therefore help to address common causes for problematic access conditions and communication requirements.

### 5.1.1 Standards for VC devices

For software access to VC devices, a few standards in the form of APIs and SDKs exist. This section will briefly introduce a number of standardized frameworks for VC systems and highlight their origin, purpose, differences and similarities before discussing in which ways frameworks like these may help address integration requirements.

**OpenVR** OpenVR is an API and SDK intended to offer an abstraction from vendor-specific VR hardware details. Is was created and is being maintained chiefly by Valve Inc.[82]
These days, OpenVR is likely the most commonly used API in the VC (thanks to the most popular IDEs and hardware devices recommending it). Valve Inc. is a member of the Open Source Virtual Reality initiative (see below, [72]), though the exact amount of cooperation is unclear.

**Open Source Virtual Reality (OSVR)** Open Source Virtual Reality is an initiative originally started by Singapore firm Razer, ostensibly to promote cooperation in the VR market by providing open source access to their hardware and software. The project's hardware and software aspects are seen as distinct subprojects.
Upon closer inspection, however, Open Source Virtual Reality may really be a bit of a misnomer. Despite their claims, it took years for Razer to finally release its hardware specifics, piecemeal at that. The 'Open Source' label was arguably meant to promote the products - chiefly the head-mounted displays - of Razer and their affiliates. Razer's headset even carries the same name. [1]

**OpenXR** OpenXR is a standard for AR and VR devices and platforms proposed by the Khrono Group: "OpenXR seeks to simplify AR/VR software development, enabling applications to reach a wider array of hardware platforms without having to port or re-write their code and subsequently allowing platform vendors supporting OpenXR access to more applications" (B.E. Insko [36]). Its version 1.0 was released at the 2019 SIGGRAPH.
OpenXR's declared aim is addressing the fragmentation of the AR and VR market for platforms and devices by introducing a common abstraction layer. Whether the solution to this lies in (yet) another standard may become apparent if and when it gains traction. The standard does, however, already seem to be in the process of being adopted by some of the large firms in the VC, including Oculus, Epic and Microsoft.

[1] On another note, a quick search of the US trademark register reveals that the acronym OSVR is even registered as a trademark by Razer (at least in the United States) under the serial numbers 86774794 for software services and 86752798 for hardware.

### 5.1.2 Shortcomings of existing standards

Two of the standards mentioned above, OpenVR and OSVR, are originally intended for VR. OpenXR is a somewhat broader standard whose developer's intent is to see it accommodate every platform and device inside the VC.
All of these standards focus on VC devices and platforms in the stricter sense; universal integration with other systems is not a priority. Nonetheless, they form a valuable basis for VC systems integration and their future extensions may well see them surpass their current narrow focus. In addition, they offer insights into how a standard for software and hardware design can be implemented, promoted and maintained.

### 5.1.3 Conclusion

While there are no standards for the VC as of yet which would provide a unified interface for the integration of interaction objects in general, some standards exist at least for the sub-set of interaction objects explicitly designed for the VC, i.e. VC peripherals and software. These standardized interfaces for interconnecting the components of distributed interaction systems can allow researchers access to systems otherwise largely opaque to them. OpenVC, OSVR and OpenXR are examples for this.
Standards also exist for some of the categories of interaction objects, such as QLM for cyber-physical objects. They have varying adoption rates but are often good starting points for integration of these objects into a VC simulation.
In terms of access conditions, the mere existence of a standard or an API does not mean that accessing it is an option in a research context. While many are advertised as 'open', this normally refers to the standard itself, not the implementation of a service behind the interface. Access to their specification also does not constitute the right to use them. Requirements detailed in the access conditions of interaction objects therefore often prohibit the use of such a service, for example for legal or privacy reasons.
The requirements which standardized interfaces help address most are communication requirements. An API for example can serve to provide clear instructions on how a service is to be used. Similarly, the physical specification of a hardware device could also be part of an open standard and permit researchers to more efficiently tailor their systems to it. Multiple systems sharing the same standardized APIs can significantly lower the effort required for their integration.

Interaction requirements can also be addressed to some degree using standards. In particular, standards for describing interaction scenarios, e.g. through Polymorphic Interaction Scenarios, can offer an abstraction layer for interaction scenarios over the specifics of implementation.
Developing and promoting standards in the VC is an important goal for researchers of the virtuality continuum. Nonetheless, the existing fragmentation between a number of competing standards should be taken into account during their development. Instead of adding to the number of standards, it may instead be possible to extend existing VC standards in order to make them compatible with more widespread systems integration instead.

## 5.2 Highly integrated development environments

Despite the increasing adoption of open standards, it seems unlikely that all firms and vendors involved will make their devices' drivers and APIs openly accessible in the near future. They will, however, need to allow their devices to be integrated with the IDEs and engines used for the development of VC simulations.[2] While these IDEs are typically proprietary and as such the user's access to the peripheral's drivers and APIs may still be limited, they may nonetheless serve to loosen the vendor lock-ins by serving as a kind of abstraction layer.
For example, peripherals such as wands may have proprietary, closed-source programming interfaces. They would, however, all need to be supported by the IDE. Even if they were not abstracted by the IDE itself, this may be done with relative ease by adding another layer using the IDE's programming support (much like was done in the Omniscope project, see appendix).
Unfortunately, this solution would still come with a number of drawbacks. First and foremost, it would be difficult to deal with any changes in the underlying layers and APIs. While it may be possible to continue using a specific version of the IDE, thus freezing its own update cycle[3], the underlying support for peripheral devices could be disrupted by firmware updates. In such a case, even if the IDE still offered support for that peripheral

[2] The simulation engines used most commonly today are bundled with integrated development environments, making IDEs and engines largely synonymous.

[3] Which is quite common, for example because Unity3D features a particularly notorious update function which requires re-downloading the entire application on each software version increment, however minor. Another reason is that the IDE often breaks vital functionality with updates, leading developers to sticking with legacy versions they believe to be more stable. That is not to say that letting one's development environment go stale is a good solution - it is merely a common one.

device's new API, there would be no guarantees that it itself may not change its own APIs in accordance with it - and the resulting change may not be compatible with the abstraction layer.
The reality of how VC game and simulation IDEs work also adds more complexity. Unity3D for example offers extensive customizability and programmability. This is useful in many ways, but as a result other hardware and software system developers choose to develop their own plug-in support for their systems (in statically or dynamically linked closed-source libraries). Support for VC systems and peripherals is therefore split across the IDE's own native support for them (as developed by the IDE's developers) and add-in frameworks (a.k.a. plugins) which are sometimes available for download in the IDE's 'Marketplace' and other times have to be obtained directly from the third-party hardware and software vendors who created them. The IDE and its plugins (and possibly even constituent parts of both) have any number of complex license and usage agreements attached to them. Unifying all of this therefore presents legal challenges alongside the many technical ones.
Since the Unity3D IDE is the IDE used for almost all VR projects in the CSTI, it will be used as an example to highlight some of the problems in using the IDE as the key system for integrating interaction objects.

### 5.2.1 Unity3D

The Unity3D IDE (or simply Unity) is one of the most commonly used IDEs for games and virtuality simulations.[77] Especially small teams, startups and researchers tend to favor it for its extensive documentation and shallow learning curve. Its strong focus on WYSIWYG features also makes it easy for people with little or no programming knowledge to contribute directly.
Unity is very extensible in a number of ways: Managed (C#) and unmanaged (C/C++) code plugins can be tied into an application quite easily[46]. Since these are integrated as compiled binaries, they often serve as a means to not just speed up crucial aspects of an application but also keep its proprietary code obfuscated, opaque and inaccessible.
Even the IDE's own look and feel can be extended using its scripting system. For example, the IDE searches all asset folders named Plugin for C# scripts at runtime and uses them to offer new menus and functions.
In addition to writing one's own plugins, Unity has a large 'Marketplace' of plugins. Comprised of assets, binaries and code, these plugins can be browsed, bought and downloaded

from within the IDE. As mentioned previously, their license models vary significantly all the way from completely free usage of open source code to monthly subscription fees for the use of closed source binaries. Some subscription models aren't immediately visible in the marketplace, as some vendors bypass Unity's payment system and instead deliver plugins which only work in conjunction to a user account, login and/or token to be obtained directly from them.

### 5.2.2 Conclusion

Highly integrated IDEs can address a number of requirements for the integration of interaction objects.
The IDE often abstracts from most of the specifics of target platforms. Provided that it itself offers sufficient access, this abstraction level can serve to address communication requirements of local systems. For example, Unity3D offers access to local file systems in a largely uniform manner regardless of the target system's operating system and file system. In a similar manner, it can serve to abstract from the details of other devices and services. Cyber-physical devices for example (such as most VC interaction devices) which do not conform to an open standard will still need to be supported by the commonly used IDEs and simulation engines. Offering support for several of them is generally achieved through an abstraction layer. In the most severe cases, use of the IDE or engine itself might be an access condition for the integration of an interaction object: Some devices may only work in conjunction with a specific IDE or engine.
On one hand, the IDE with the plugins and extensions available for it can provide easy integration of many potential objects of interaction: Remote services like speech-to-text services, multimedia players and streaming software, video analysis services can all be available out of the box or as plugin or extension. On the other hand, their use may not be possible due to the cost of their use as well as inadequate privacy assurances or security. Their opacity can be another concern, as the proprietary nature of many implementations can make them difficult to understand, much less configure. The inadequacies and opacity of the multimedia player embedded in the Unity3D engine was a contributing factor in the development of the Omniscope project (see [46]). It therefore varies whether the IDE helps or hinders access conditions of interaction objects - there is certainly potential for both.
Similarly, some communication requirements can become a non-issue thanks to the entire implementation already being provided. Others, on the other hand, can become even

more problematic: Since the implementation of the integration is only partially under the control of the researchers, some requirements may be difficult to address. For example, it may not be possible to change the way on which a plugin communicates with a web service from a synchronous/blocking to an asynchronous behaviour.
Implementations of interaction techniques can also be available as part of the IDE, engine or plugin system, though even when they are, tailoring them to a specific research context and use case can still be a significant effort.

## 5.3 Inter-system communication and integration platforms

Whereas VC product vendors have started to agree on open standards and APIs, the market for smart environments and the IoT is still very fragmented. With larger firms all trying to push their own vendor-specific solutions, inter-operability on the basis of common standards is rarely an option.
Instead, the distributed nature of cyber-physical and VC systems offers two other main avenues for integration: Inter-system communication frameworks and system integration platforms.

### 5.3.1 Inter-system communications frameworks and middlewares

Inter-system communication frameworks is used here as a term encompassing those networking and messaging systems used for the exchange of data between constituent parts of a distributed system. Some exist in the form of standards and protocols and may have many implementations to choose from (both open source and proprietary), others are strictly closed-source products.
As opposed to the fully-fledged system integration platforms mentioned in the next section, communications frameworks are leaner in that they only deal with the actual exchange of information between subsystems rather than dictating much of the architecture of those subsystems. Perhaps the most commonly used type of framework in the IoT is that of messaging systems, e.g. publish/subscribe systems like MQTT.
Messaging systems are particularly important for actor systems (systems based on the actor model) as they form a basis for them. Systems developed in accordance with actor model principles continue to be quite popular with developers of complex distributed systems. Whether they are a good fit for distributed interaction systems depends on the

kind of system to be integrated.
As an example for a standard/protocol based messaging framework, the MQTT standard will be discussed. MQTT was applied in the Shelldon and VibeGlove projects. In contrast to this, the closed-source CSTI Middleware will also be shown. The CSTI middleware is based on the Akka actor framework.

### Messaging Systems: MQTT

MQTT (Message Queueing Telemetry Transport) is a standard for a publish/subscribe communications protocol. It has been approved by the ISO as standard ISO/IEC PRF 20922 (see [31]).
The MQTT Standard has been very widely adopted and forms the basis for many messaging-based communications systems. There are also different variants and offspawns of this standard, including MQTT-SN for non-TCP/IP-based communications such as may be found in some cyber-physical networks.
Due to its wide-spread adoption and maturity (the MQTT standard was first proposed in 1999, making it around two decades old), MQTT serves as a good example for messaging-based inter-systems communication. It has also been used for a number of projects within the CSTI, such as in the Shelldon prototypes experiment: The bale of smart turtles communicates with one another and the host station by publishing and subscribing to channels of an MQTT server on their host station (a Mosquitto server [18]).
Messaging systems like MQTT can serve to facilitate much of the inter-systems communication needed for the integration of different interaction objects. Acting as a common point of reference, they may aid in the design of interaction systems components: These components would not need to know much about the specifics of their communication counterparts aside from an agreed upon taxonomy and format of messages and their content.
Publish/Subscribe systems in particular are useful in permitting loosely coupled communications which do not require the different services to be in direct contact with one another. In essence, they can offer basic discovery services. This can help make the constituent parts of a distributed interaction system more resilient to changes, such as unavailable services. An example for this can be seen in the Shelldon project.
They also and especially lend themselves to loosely coupled event-driven approaches. For example, within a VC simulation, an MQTT system could offer channels for the user's limbs as they are being monitored by motion tracking systems. Events would then be

published to these channels whenever the user's limbs were to touch a simulated object. Haptic/Tactile feedback systems such as microcontrollers and vibration motors in the user's clothing could subscribe to these channels, triggering direct physical stimuli whenever such a contact occurred.
The benefits to the use of a centralized messaging system do not come without drawbacks. Of particular note is the possibility of catastrophic failure of all those parts of a distributed interaction system that depend upon it if this central component itself should fail. Precautions should be taken in an effort to prevent this, such as a redundant network of communication servers.
Another drawback lies in the possibility of violating requirements by relaying data through the server. This is likely to cause increased latency at the very least, possibly accompanied by other quality-of-service problems, such as messages being lost, duplicated or subsequent messages arriving in an incorrect order. This is not unlikely to violate communication requirements. Some messaging systems offer safeguards to alleviate some of these problems. MQTT for example offers three QOS levels: 0, 1 and 2. These QoS levels guarantee the delivery of any given message 'at most once', 'at least once' and 'exactly once', respectively. These precautions do not come without their own cost, with higher QoS guarantees causing more data to be transferred between parties and latencies increasing as more handshakes, re-transmissions and confirmations are required.
The messaging system's own hosting can also be a concern - it is essentially a software system (local or remote) integrated into the VC system. Access conditions apply, including data protection concerns if it is hosted remotely.
In conclusion, messaging services exist in many varieties and in their configuration may offer a number of trade-offs, allowing them to be customized for a wide range of use cases. They are well suited as a communications component for many interaction systems, but not for all of them.

### CSTI Middleware

The CSTI Middleware (see [19]) is a platform intended to make the integration of software systems in a laboratory context as transparent, scalable and maintainable as possible. The motivations behind its use in the research context includes typical problem in a university laboratory in which the primary workforce is comprised of students: Any work done by any one of them will - sooner or later - cease, typically after their thesis or project has run its course. Maintaining the results of that work without the original con-

tributor(s) and using it as the foundation for further research work can be prohibitively difficult if its interfaces are opaque, undocumented and unstandardized. To this end, the CSTI Middleware serves as a unifying inter-systems communications solution which students are typically obliged to integrate their projects with.
The CSTI Middleware is based on the Akka actor framework and written in Scala. It also leverages Protobuf for documented and efficient serialization/deserialization. The JSON structures exchanged through it are versioned in the lab's version control systems, typically alongside the projects they stem from.
The Middleware turned out to improve some of the interoperability of the systems inside the laboratory. It therefore presents an improvement at least over having no commonly agreed upon way to exchange messages between systems at all. There were concerns, however, particularly in three key areas: Its fitness for specific use-cases (it being a rather generic solution), its performance and its maintenance.
Its fitness for specific use-cases was limited first and foremost by the fact that it was based purely on exchanging JSON style objects. This made it a bad fit for use cases such as transferring large amounts of binary data. In addition, it works as a publish/subscribe system which does not suit some use cases like multimedia streaming. Such drawbacks may be addressed by accompanying the Middleware with dedicated systems for such use cases and using the Middleware for control and configuration. For example, the Middleware could be used to advertise streaming services over the network, with the actual streaming being handled using another system.
Its performance was also a point of contention. When used purely as a publish/subscribe messaging system it seemed to perform slower than Mosquitto.
The Middleware project was maintained by a single developer and subject to API-breaking changes with little or no notice.
None of these problems are opposed to the proposed benefits of the Middleware for its central purposes. They may however limit its usefulness for the purposes of acting as a basis for a distributed interaction system.
A sensible next step would be to quantitatively compare the Middleware to alternatives in a variety of use cases. It would also be necessary to try and separate the technical solutions with its benefits and drawbacks from the other constituent parts such as making sure developers document and offer their APIs in an organized fashion. This could take the form of a control study.

### 5.3.2 System integration platforms

In the previous section it was shown that communications frameworks can serve not just to connect different systems but also as a unifying factor in interface design and standardization. One shortcoming is the integration of third-party systems that developers do not have source access or rights to or enough knowledge of - such systems can not be integrated without additional connectors or wrappers.
System integration platforms are typically much larger and more complex systems than simple communications solutions. They integrate communication frameworks and extend them with adapters, wrappers, connectors and bindings.
There is a large overlap between unifying standards like those mentioned in the beginning of this chapter and these system integration platforms. For the purpose of this thesis, the distinction between the two is that standards - while providing a commonly agreed upon way of implementing a system - do not provide much of the actual implementation and are not capable of running on their own. System integration platforms on the other hand consist of software (and possibly hardware) facilitating integrations of systems - often to adapt all of them to the same API or standard. OpenHAB is a common example for this and often used in research contexts.

#### OpenHAB

In the field of smart home systems, the domestic side of smart objects and the IoT, OpenHAB has established itself as one of the most comprehensive open source offerings for integrating various devices and systems. It mainly owes its support for a very extensive range of products to its equally extensive community of volunteer contributors. The maker scene in particular is a pillar of its popularity and growth, endeavoring to enable the integration of any and all home automation system with the platform. It has also been used extensively in research contexts, providing the basis for many prototypes, demonstrations and experiments in the fields of cyber-physical systems, home automation and the IoT. The adaptation of QLM for the IoT mentioned before (proposed in [71]) for example also uses OpenHAB to demonstrate their framework's practicability.
OpenHAB can serve as a good example of how powerful an open platform for systems integration can become. It may even be of use to researchers directly - integrating it with a VC simulation can offer access to domestic smart environments with a minimum of effort.

OpenHAB's architecture is clearly oriented at domestic automation and it may therefore not suit other use-cases of VC applications. It can nonetheless provide a template for how an open platform can serve to effectively integrate any number of systems.
Their own integration could come with its own integration requirements. Rule-based automation engines and If-This-Then-That (ITTT) mechanisms like those of a home automation system can be leveraged from within the VC application in different ways. An example of how VC interaction can be used to define such rules can be found in [61].

### 5.3.3 Conclusion

Inter-system communication between integrated sub-systems can aid with communications requirements, offering an efficient, unified and well-documented basis for information interchange. That said, for this to happen there first has to be the option of letting the interaction object communicate using it: A lot of them will not allow for any form of communication other than what has been provided by their developers and vendors. This is where integration platforms like OpenHAB can come in: integrating many different systems using wrappers, bindings and a unified API.
With the use of an integration platform, much of the heavy lifting in terms of integration implementations has been done and many of the requirements addressed. However, some of the requirements specific to interaction objects remain.
Access conditions are defined not only by the integration platform itself (and who is operating and hosting it) but also of all the systems interconnected with it. While the platform could be safe, efficient and open, connected services may not be. Even with the integration being offered on a silver platter, it is still necessary to carefully examine all the technical, legal and ethical conditions of their use.
Unified platforms like these are also somewhat prone to causing technological fragmentation and vendor lock-ins (see [7]).
If the platform is based on open source code and specifications like OpenHAB, many of these concerns are relatively easily overcome: It can be adjusted, customized and hosted to the exacting specifications of the research context. Much the same can be said of the performance of the implementations they provide: If insufficient, it could be adapted. That said, this can be a significant effort and may therefore not be possible for some research contexts. In addition, the ongoing maintenance of a communication or integration solution instance can also be a drain on resources. This should enter into a research

team's consideration about whether or not to use them.
Another issue which current platforms do not yet address is interaction design for the VC. This is the main aspect which should set apart future integration platforms that have been designed specifically for the integration of interaction objects. Nonetheless, it might already be possible to develop interaction designs for the APIs of other platforms like OpenHAB in a way which efficiently exposes their functionality inside a VC applicaton, as it has been done in [61].
In summary, an inter-systems communications solution or even an entire integration platform can form a useful layer for unifying systems as they are being integrated into the VC. They can address many but likely not all of the requirements for the integration of interaction objects. Some platforms already exist for fields such as home automation, but none explicitly for the VC as of yet - this would be a useful avenue for further research.

## 5.4 Conclusion

Human-Computer Interaction in the virtuality continuum is a field currently in rapid flux. Development strategies and tooling support for applications within the VC are therefore lacking in many areas, including their suitability for integrating new objects of interaction. The integration of interaction objects is an arduous task that often needs to be repeated for each of many isolated, insular VC experiences. While there is a wide spectrum of interaction objects which could be integrated, there is currently no solution which would address the majority of their integration requirements. Even if there were such a panacea for technical requirements, it would still have to avoid non-technical requirements such as being ethical, secure and open to modification in order to be suitable for research contexts.
Nonetheless, while there may not be a one-size-fits-all solution, it has been shown that existing technological and methodological solutions exist for many of the requirements posed by the integration of interaction objects. Standards, integrated development environments and dedicated inter-system communication and integration solutions all address a significant share of the requirements. A basis for a distributed interaction system could therefore possibly be created by combining them.
This chapter's analysis of strategies and technologies has demonstrated that the requirements catalogue proposed in this thesis can work as a basis for discussion of existing solutions.

# 6 Summary and Future Works

## 6.1 Summary

This thesis introduced four categories for interaction objects which could be integrated into distributed interaction systems: virtual periphery systems, multimedia/streaming systems, cyber-physical systems and unconnected/'dumb' objects. It proposed three types of integration requirements for these objects: access conditions, communication requirements and interaction requirements. It then laid out and highlighted characteristics of potential interaction objects in each of their categories with examples. It outlined their working principles and showcased how they could be integrated into a VC application, cataloguing the requirements their integration would pose. It also raised concerns with regards to ethics and privacy in VC applications in general and those of a distributed nature in particular.
It discussed some of the peculiarities of the development of VC applications in research contexts and their relationship with (and differences to) the development of digital games. The concept of task-oriented development as an antithesis of the common pursuit of realistic and immersive simulations at the cost of their practical purpose was introduced. As an instance of task-oriented development, the development of economical VC applications in particular was proposed: a guiding principle for applications developed in the CSTI today.
A number of potential strategies and technologies for easing and organizing the integration of distributed interaction objects were introduced: Open standards, integrated development environments and inter-systems communication and integration solutions. They were compared to the integration requirements and discussed with regards to their benefits and drawbacks. Each has the potential to address some of the integration requirements of interaction objects. Their combination may serve as a good starting point for the development of a distributed interaction system.
The results of this thesis can serve researchers in two main ways: Firstly, as a toolbox

of practical advice for how distributed interaction objects can be integrated into a VC system in order to extend the range of available interaction scenarios. Secondly, by providing means to discuss and evaluate strategies and technologies which aim to make such integrations easier and address their requirements.

## 6.2 Future Works and Prospects

The exploration of the VC's potential in changing the way humans interact with computers has only just begun. Based on this thesis, the development of methodologies and technologies for the facilitation and orchestration of distributed interaction scenarios may unlock more of its potential. This may help pave some of the way toward the 'Ultimate Simulation' - or at least improve the potential for VC applications to be used productively in our daily lives.
This thesis' findings are only a first indication of the requirements posed by the integration of distributed interaction scenarios based on contemporary literature and experiments, further research will be required to fully validate them. Within research contexts, that process could accompany the ongoing development of interactive VC simulations by other researchers. They could be used as case studies as well as testing grounds for technical solutions and development methodologies.
In particular, it would be important to compare different strategies and technologies not only with regards to their feasibility and the problems they may serve to solve but also with regards to their ease of use and adoption.
Extending the proposed catalogue of requirements should be another process accompanying the ongoing work in research contexts. It is probable that the integration of services and devices not considered in this thesis - and quite possibly as of yet undeveloped - will necessitate additions to this list and possible to the proposed categories of interaction objects.
Research and trial of task-oriented development principles in general and economical VC applications in particular will continue.
The further development of the Polymorphic Interaction Scenarios which this thesis proposed as a means of unifying the way in which interaction scenarios are developed, specified and/or documented is also ongoing. They may eventually become the basis of a standard for the specification of interaction techniques.
It seems certain that the integration of distributed systems for the purposes of offering new interaction scenarios will progress significantly in the near future. The results of this

thesis can aid in their development and evaluation.
If systems in the virtuality continuum continue to be enhanced in ways which allow us to seamlessly interact between real and virtual worlds, perhaps it will soon be necessary to re-evaluate what 'reality' truly means to us.

# Bibliography


[1] A-Frame: *A-Frame VR Web-Framework (official website)*. 2019. – URL https://aframe.io/. – Zugriffsdatum: 2019-08-08

[2] Andersen, B. J. H. ; Davis, A. T. A. ; Weber, G. ; Wünsche, B. C.: Immersion or Diversion: Does Virtual Reality Make Data Visualisation More Effective? In: *2019 International Conference on Electronics, Information, and Communication (ICEIC)*, Jan 2019, S. 1–7

[3] Baum, Dr. N. ; Uhlig, Markus: *Pattarina Sewing Pattern Application*. 2019. – URL https://www.pattarina.de/. – Zugriffsdatum: 2019-08-08

[4] Belloch, Jose A. ; Gonzalez, Alberto ; Quintana-Orti, Enrique S. ; Ferrer, Miguel ; Valimaki, Vesa ; Belloch, Jose A. ; Gonzalez, Alberto ; Quintana-Orti, Enrique S. ; Ferrer, Miguel ; Valimaki, Vesa: GPU-Based Dynamic Wave Field Synthesis Using Fractional Delay Filters and Room Compensation. In: *IEEE/ACM Trans. Audio, Speech and Lang. Proc.* 25 (2017), Februar, Nr. 2, S. 435–447. – URL https://doi.org/10.1109/TASLP.2016.2631338. – Zugriffsdatum: 2019-08-08. – ISSN 2329-9290

[5] Bleich, Holger: *Amazon Reveals Private Voice Data Files*. 2019. – URL https://www.heise.de/newsticker/meldung/Amazon-reveals-private-voice-data-files-4256015.html. – Zugriffsdatum: 2019-08-08

[6] Bronsch, Johann: *Einsatz von Virtual-Reality Techniken in Lernumgebungen*, HAW Hamburg, Diplomarbeit, 2018. – URL https://kataloge.uni-hamburg.de/DB=2/XMLPRS=N/PPN?PPN=1022279033. – Zugriffsdatum: 2019-08-08

[7] Brown, William ; Malveau, Raphael ; McCormick, Hays ; Mowbray, Thomas ; Thomas., Scott W.: *Antipatterns: Vendor Lock-In*. 2017. – URL http://antipatterns.com/vendorlockin.htm. – Zugriffsdatum: 2019-08-08

[8] Buyya, Rajkumar ; Broberg, James ; Goscinski, Andrzej M.: *Cloud Computing Principles and Paradigms*. Wiley Publishing, 2011. – ISBN 9780470887998

[9] Casas, Llogari ; Ciccone, Loïc ; Çimen, Gökçen ; Wiedemann, Pablo ; Fauconneau, Matthias ; Sumner, Robert W. ; Mitchell, Kenny: Multi-reality Games: An Experience Across the Entire Reality-virtuality Continuum. In: *Proceedings of the 16th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry*. New York, NY, USA : ACM, 2018 (VRCAI '18), S. 18:1–18:4. – URL http://doi.acm.org/10.1145/3284398.3284411. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-4503-6087-6

[10] Center for Speech Technology Research at the University of Edinburgh: *The Festival Speech Synthesis System*. 2019. – URL http://www.cstr.ed.ac.uk/projects/festival/. – Zugriffsdatum: 2019-08-08

[11] Chen, Wen ; Sharieh, Salah ; Blainey, Bob: A Security-as-a-service Solution for Applications in Cloud Computing Environment. In: *Proceedings of the Communications and Networking Symposium*. San Diego, CA, USA : Society for Computer Simulation International, 2018 (CNS '18), S. 4:1–4:9. – URL http://dl.acm.org/citation.cfm?id=3213200.3213204. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-5108-6015-5

[12] Chen, Yang-Sheng ; Han, Ping-Hsuan ; Lee, Kong-Chang ; Hsieh, Chiao-En ; Hsiao, Jui-Chun ; Hsu, Che-Ju ; Chen, Kuan-Wen ; Chou, Chien-Hsing ; Hung, Yi-Ping: Lotus: Enhancing the Immersive Experience in Virtual Environment with Mist-based Olfactory Display. In: *SIGGRAPH Asia 2018 Virtual & Augmented Reality*. New York, NY, USA : ACM, 2018 (SA '18), S. 9:1–9:2. – URL http://doi.acm.org/10.1145/3275495.3275503. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-4503-6028-9

[13] Cruz-Neira, Carolina ; Sandin, Daniel J. ; DeFanti, Thomas A.: Surround-screen Projection-based Virtual Reality: The Design and Implementation of the CAVE. In: *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA : ACM, 1993 (SIGGRAPH '93), S. 135–142. – URL http://doi.acm.org/10.1145/166117.166134. – Zugriffsdatum: 2019-08-08. – ISBN 0-89791-601-8

[14] Cumming, Geoff: QWERTY and keyboard reform: the soft keyboard option. In: *International Journal of Man-Machine Studies* 21 (1984), Nr. 5, S. 445

– 450. – URL http://www.sciencedirect.com/science/article/pii/S002073738480070X. – Zugriffsdatum: 2019-08-08. – ISSN 0020-7373

[15] Donalek, C. ; Djorgovski, S. G. ; Cioc, A. ; Wang, A. ; Zhang, J. ; Lawler, E. ; Yeh, S. ; Mahabal, A. ; Graham, M. ; Drake, A. ; Davidoff, S. ; Norris, J. S. ; Longo, G.: Immersive and collaborative data visualization using virtual reality platforms. In: *2014 IEEE International Conference on Big Data (Big Data)*, Oct 2014, S. 609–614

[16] Dr. Eva Geisberger, Prof. Dr. Dr. h. c. Manfred B.: *agendaCPS: Integrierte Forschungsagenda Cyber-Physical Systems*. 1st. acatech, 2012. – URL https://www.bmbf.de/files/acatech_STUDIE_agendaCPS_Web_20120312_superfinal.pdf. – Zugriffsdatum: 2019-08-08

[17] Dörner, Ralf ; Broll, Wolfgang ; Grimm, Paul ; Jung, Bernhard: *Virtual und Augmented Reality (VR / AR)*. Springer, 2013. – URL https://link.springer.com/book/10.1007%2F978-3-642-28903-3. – Zugriffsdatum: 2019-08-08

[18] Eclipse Foundation: *Eclipse Mosquitto MQTT-Broker*. 2019. – URL https://mosquitto.org/. – Zugriffsdatum: 2019-08-08

[19] Eichler, Tobias ; Draheim, Susanne ; Grecos, Christos ; Wang, Qi ; von Luck, Kai: Scalable Context-Aware Development Infrastructure for Interactive Systems in Smart Environments. In: *Fifth International Workshop on Pervasive and Context-Aware Middleware 2017 (PerCAM'17)*. Rome, Italy, Oktober 2017

[20] Erickson, Thomas: Artificial Realities as Data Visualization Environments: Problems and Prospects. In: Wexelblat, Alan (Hrsg.): *Virtual Reality*. Academic Press, 1993, S. 3 – 22. – URL http://www.sciencedirect.com/science/article/pii/B978012745045250009X. – Zugriffsdatum: 2019-08-08. – ISBN 978-0-12-745045-2

[21] Ernst, M. O. ; Banks, M. S.: Humans integrate visual and haptic information in a statistically optimal fashion. In: *Nature* 415 (2002), Januar, S. 429–433. – URL https://ui.adsabs.harvard.edu/abs/2002Natur.415..429E. – Zugriffsdatum: 2019-08-08. – Provided by the SAO/NASA Astrophysics Data System

[22] Ernst, Marc O.: Multisensory Integration: A Late Bloomer. In: *Current Biology* 18 (2008), Nr. 12, S. R519 – R521. – URL http://www.sciencedirect.com/science/article/pii/S0960982208005976. – Zugriffsdatum: 2019-08-08. – ISSN 0960-9822

[23] Främling, Kary ; Maharjan, Merina: Standardized Communication Between Intelligent Products for the IoT*. In: *IFAC Proceedings Volumes* 46 (2013), Nr. 7, S. 157 – 162. – URL http://www.sciencedirect.com/science/article/pii/S1474667015356676. – Zugriffsdatum: 2019-08-08. – 11th IFAC Workshop on Intelligent Manufacturing Systems. – ISSN 1474-6670

[24] FSF: *Categories of free and nonfree software*. 2019. – URL https://www.gnu.org/philosophy/categories.html.en. – Zugriffsdatum: 2019-08-08

[25] FSF: *The Free Software Definition*. 2019. – URL https://www.gnu.org/philosophy/free-sw.html. – Zugriffsdatum: 2019-08-08

[26] Geens, Koen: *Belgian Ban on Loot Boxes (Statement by the Belgian Minister of Justice)*. 2019. – URL https://www.koengeens.be/news/2018/04/25/loot-boxen-in-drie-videogames-in-strijd-met-kansspelwetgeving. – Zugriffsdatum: 2019-08-08

[27] Grigore C. Burdea, Philippe C.: *Virtual Reality Technology, 2nd Edition*. Wiley, 2003. – URL https://www.wiley.com/en-us/Virtual+Reality+Technology%2C+2nd+Edition-p-9780471360896. – Zugriffsdatum: 2019-08-08. – ISBN 978-0-471-36089-6

[28] Grigore C. Burdea, Philippe C.: *Understanding Virtual Reality: Interface, Application, and Design, Second Edition*. Elsevier, 2018. – ISBN 978-0-12-800965-9

[29] Hannun, Awni ; Case, Carl ; Casper, Jared ; Catanzaro, Bryan ; Diamos, Greg ; Elsen, Erich ; Prenger, Ryan ; Satheesh, Sanjeev ; Sengupta, Shubho ; Coates, Adam ; Y. Ng, Andrew: DeepSpeech: Scaling up end-to-end speech recognition. (2014), 12

[30] Hecomi: *uDesktopDuplication*. 2019. – URL https://github.com/hecomi/uDesktopDuplication. – Zugriffsdatum: 2019-08-08

[31] Information technology - Message Queuing Telemetry Transport (MQTT) / International Organization for Standardization. Geneva, CH, Juni 2016. – Standard.

– URL https://www.iso.org/standard/69466.html,urldate={2019-08-08},

[32] Jacob, Robert J. ; Girouard, Audrey ; Hirshfield, Leanne M. ; Horn, Michael S. ; Shaer, Orit ; Solovey, Erin T. ; Zigelbaum, Jamie: Reality-based Interaction: A Framework for post-WIMP Interfaces. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM, 2008 (CHI '08), S. 201–210. – URL http://doi.acm.org/10.1145/1357054.1357089. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-60558-011-1

[33] Jo, Dongsik ; Kim, Yongwan ; Jeon, Woojin ; Kim, Yongsun ; Kim, Hongki ; Kim, Ki-Hong ; Kwak, Seungho: VR Planning Toolkit to Simulate Physical and Virtual Configurations: A Case Study of an Indoor VR Roller Coaster Augmenting Experience. In: *Proceedings of the 9th Augmented Human International Conference*. New York, NY, USA : ACM, 2018 (AH '18), S. 27:1–27:3. – URL http://doi.acm.org/10.1145/3174910.3174912. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-4503-5415-8

[34] Jude, Alvin ; Poor, G. M. ; Guinness, Darren: Grasp, Grab or Pinch? Identifying User Preference for In-Air Gestural Manipulation. In: *Proceedings of the 2016 Symposium on Spatial User Interaction*. New York, NY, USA : ACM, 2016 (SUI '16), S. 219–219. – URL http://doi.acm.org/10.1145/2983310.2989209. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-4503-4068-7

[35] Kato, Shingo ; Nakamoto, Takamichi: Demo of Olfactory Display with Less Residual Odor. In: *SIGGRAPH Asia 2018 Emerging Technologies*. New York, NY, USA : ACM, 2018 (SA '18), S. 1:1–1:2. – URL http://doi.acm.org/10.1145/3275476.3275493. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-4503-6027-2

[36] Khronos Group: *OpenXR*. 2019. – URL https://www.khronos.org/openxr. – Zugriffsdatum: 2019-08-08

[37] Knierim, Pascal ; Kosch, Thomas ; Schwind, Valentin ; Funk, Markus ; Kiss, Francisco ; Schneegass, Stefan ; Henze, Niels: Tactile Drones - Providing Immersive Tactile Feedback in Virtual Reality Through Quadcopters. In: *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA : ACM, 2017 (CHI EA '17), S. 433–436. – URL http://doi.acm.org/10.1145/3027063.3050426. – ISBN 978-1-4503-4656-6

[38] Kreylos, Oliver: *Lighthouse tracking examined*. 2019. – URL http://doc-ok.org/?p=1478. – Zugriffsdatum: 2019-08-08

[39] Lederman, S. J. ; Klatzky, R. L.: Haptic perception: A tutorial. In: *Attention, Perception, & Psychophysics* 71 (2009), Oct, Nr. 7, S. 1439–1459. – URL https://doi.org/10.3758/APP.71.7.1439. – Zugriffsdatum: 2019-08-08. – ISSN 1943-393X

[40] Leng, Hoo Y. ; Norowi, Noris M. ; Jantan, Azrul H.: A User-Defined Gesture Set for Music Interaction in Immersive Virtual Environment. In: *Proceedings of the 3rd International Conference on Human-Computer Interaction and User Experience in Indonesia*. New York, NY, USA : ACM, 2017 (CHIuXiD '17), S. 44–51. – URL http://doi.acm.org/10.1145/3077343.3077348. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-4503-5212-3

[41] Löchtefeld, Markus ; Lappalainen, Tuomas ; Väyrynen, Jani ; Colley, Ashley ; Häkkilä, Jonna: Comparing Thermal and Haptic Feedback Mechanisms for Game Controllers. In: *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. New York, NY, USA : ACM, 2017 (CHI EA '17), S. 1829–1836. – URL http://doi.acm.org/10.1145/3027063.3053172. – ISBN 978-1-4503-4656-6

[42] Marks, Stefan ; Estevez, Javier E. ; Connor, Andy M.: Towards the Holodeck: Fully Immersive Virtual Reality Visualisation of Scientific and Engineering Data. In: *Proceedings of the 29th International Conference on Image and Vision Computing New Zealand*. New York, NY, USA : ACM, 2014 (IVCNZ '14), S. 42–47. – URL http://doi.acm.org/10.1145/2683405.2683424. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-4503-3184-5

[43] McGraw, I. ; Prabhavalkar, R. ; Alvarez, R. ; Arenas, M. G. ; Rao, K. ; Rybach, D. ; Alsharif, O. ; Sak, H. ; Gruenstein, A. ; Beaufays, F. ; Parada, C.: Personalized speech recognition on mobile devices. In: *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2016, S. 5955–5959. – ISSN 2379-190X

[44] Melles, Gerald: *Interaktionen in Virtuellen Welten: Ein Testframework*. 2016. – URL https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2016-hsem/melles/bericht.pdf. – Zugriffsdatum: 2019-08-08

[45] Melles, Gerald: *A taxonomic basis for Virtual Reality technologies (Grundprojekt)*. 2017. – URL https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2017-proj/melles.pdf. – Zugriffsdatum: 2019-08-08

[46] Melles, Gerald: The Omniscope - Multimedia Streaming and Computer Vision for Applications in the Virtuality Continuum. In: *SKILL 2018 Studierendenkonferenz Informatik*, Sep 2018, S. 49–54

[47] Memory Alpha: *Holodeck (Star Trek fanwiki entry)*. 2019. – URL https://memory-alpha.fandom.com/wiki/Holodeck. – Zugriffsdatum: 2019-08-08

[48] Memory Alpha: *Replicator (Star Trek fanwiki entry)*. 2019. – URL https://memory-alpha.fandom.com/wiki/Replicator. – Zugriffsdatum: 2019-08-08

[49] Merriam Webster Online Dictionary: *Definition of Illusion*. 2019. – URL https://www.merriam-webster.com/dictionary/illusion. – Zugriffsdatum: 2019-08-08

[50] Merriam Webster Online Dictionary: *Definition of Standard*. 2019. – URL https://www.merriam-webster.com/dictionary/standard. – Zugriffsdatum: 2019-08-08

[51] Milgram, Paul ; Kishino, Fumio: A Taxonomy of Mixed Reality Visual Displays. vol. E77-D, no. 12 (1994), 12, S. 1321–1329. – URL http://etclab.mie.utoronto.ca/people/paul_dir/IEICE94/ieice.html. – Zugriffsdatum: 2019-08-08

[52] Mozilla: *Firefox Reality Browser*. 2019. – URL https://mixedreality.mozilla.org/firefox-reality/. – Zugriffsdatum: 2019-08-08

[53] Mycroft AI Inc.: *Mycroft: Fallback Skills*. 2019. – URL https://mycroft.ai/documentation/skills/fallback-skill/. – Zugriffsdatum: 2019-08-08

[54] Nakamura, Hiromi ; Miyashita, Homei: Augmented Gustation Using Electricity. In: *Proceedings of the 2Nd Augmented Human International Conference*. New York, NY, USA : ACM, 2011 (AH '11), S. 34:1–34:2. – URL http://doi.acm.org/10.1145/1959826.1959860. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-4503-0426-9

[55] Narumi, Takuji ; Nishizaka, Shinya ; Kajinami, Takashi ; Tanikawa, Tomohiro ; Hirose, Michitaka: Augmented Reality Flavors: Gustatory Display Based on Edible Marker and Cross-modal Interaction. In: *Proceedings of the SIGCHI Conference*

*on Human Factors in Computing Systems*. New York, NY, USA : ACM, 2011 (CHI '11), S. 93–102. – URL http://doi.acm.org/10.1145/1978942.1978957. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-4503-0228-9

[56] Noyes, Jan: The QWERTY keyboard: a review. In: *International Journal of Man-Machine Studies* 18 (1983), Nr. 3, S. 265 – 281. – URL http://www.sciencedirect.com/science/article/pii/S0020737383800108. – Zugriffsdatum: 2019-08-08. – ISSN 0020-7373

[57] OGRE: *OGRE Open Source 3D Graphics Engine*. 2019. – URL https://www.ogre3d.org/. – Zugriffsdatum: 2019-08-08

[58] OpenJS Foundation: *Node-RED Framework*. 2019. – URL https://nodered.org/. – Zugriffsdatum: 2019-08-08

[59] OSI: *The Open Source Definition*. 2019. – URL https://opensource.org/osd. – Zugriffsdatum: 2019-08-08

[60] O'Toole, Alice J. ; Phillips, P. J. ; An, Xiaobo ; Dunlop, Joseph: Demographic effects on estimates of automatic face recognition performance. In: *Image and Vision Computing* 30 (2012), Nr. 3, S. 169 – 176. – URL http://www.sciencedirect.com/science/article/pii/S0262885612000029. – Zugriffsdatum: 2019-08-08. – Best of Automatic Face and Gesture Recognition 2011. – ISSN 0262-8856

[61] Park, Heonjin ; Koh, Kyle ; Choi, Yuri ; Chae, Han J. ; Hwang, Jeongin ; Seo, Jinwook: Defining Rules Among Devices in Smart Environment Using an Augmented Reality Headset. In: *Proceedings of the Second International Conference on IoT in Urban Space*. New York, NY, USA : ACM, 2016 (Urb-IoT '16), S. 18–21. – URL http://doi.acm.org/10.1145/2962735.2962746. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-4503-4204-9

[62] Peiris, Roshan L. ; Peng, Wei ; Chen, Zikun ; Chan, Liwei ; Minamizawa, Kouta: ThermoVR: Exploring Integrated Thermal Haptic Feedback with Head Mounted Displays. In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM, 2017 (CHI '17), S. 5452–5456. – URL http://doi.acm.org/10.1145/3025453.3025824. – ISBN 978-1-4503-4655-9

[63] Phillips, P. J. ; Jiang, Fang ; Narvekar, Abhijit ; Ayyad, Julianne ; O'Toole, Alice J.: An Other-race Effect for Face Recognition Algorithms. In: *ACM Trans. Appl. Percept.* 8 (2011), Februar, Nr. 2, S. 14:1–14:11. – URL http://doi.acm.org/10.1145/1870076.1870082. – Zugriffsdatum: 2019-08-08. – ISSN 1544-3558

[64] pmoews: *Garden Turtle (3D Scan/Model).* 2014. – URL https://www.thingiverse.com/thing:421809. – Zugriffsdatum: 2019-08-08

[65] Prattichizzo, Domenico ; Pacchierotti, Claudio ; Cenci, Stefano ; Minamizawa, Kouta ; Rosati, Giulio: Using a Fingertip Tactile Device to Substitute Kinesthetic Feedback in Haptic Interaction. In: Kappers, Astrid M. L. (Hrsg.) ; Erp, Jan B. F. van (Hrsg.) ; Bergmann Tiest, Wouter M. (Hrsg.) ; Helm, Frans C. T. van der (Hrsg.): *Haptics: Generating and Perceiving Tangible Sensations.* Berlin, Heidelberg : Springer Berlin Heidelberg, 2010, S. 125–130. – URL http://sirslab.dii.unisi.it/papers/2010/Prattichizzo.EUROHAPTICS.2010.Haptics.Pub.pdf. – Zugriffsdatum: 2019-08-08

[66] Romero-Rondón, Miguel F. ; Sassatelli, Lucile ; Precioso, Frédéric ; Aparicio-Pardo, Ramon: Foveated Streaming of Virtual Reality Videos. In: *Proceedings of the 9th ACM Multimedia Systems Conference.* New York, NY, USA : ACM, 2018 (MMSys '18), S. 494–497. – URL http://doi.acm.org/10.1145/3204949.3208114. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-4503-5192-8

[67] Ryoo, Jihoon ; Yun, Kiwon ; Samaras, Dimitris ; Das, Samir R. ; Zelinsky, Gregory: Design and Evaluation of a Foveated Video Streaming Service for Commodity Client Devices. In: *Proceedings of the 7th International Conference on Multimedia Systems.* New York, NY, USA : ACM, 2016 (MMSys '16), S. 6:1–6:11. – URL http://doi.acm.org/10.1145/2910017.2910592. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-4503-4297-1

[68] Saddik, Abdulmotaleb E. ; Orozco, Mauricio ; Eid, Mohamad ; Cha, Jongeun: *Haptics Technologies: Bringing Touch to Multimedia.* 1st. Springer Publishing Company, Incorporated, 2011. – ISBN 3642226574, 9783642226571

[69] Schell, Jesse: *The Art of Game Design: A Book of Lenses.* 2nd. CRC Press, 2014. – ISBN 9781466598645

[70] SFUptownMaker: *Leap Motion Teardown*. 2017. – URL https://learn.sparkfun.com/tutorials/leap-motion-teardown. – Zugriffsdatum: 2019-08-08

[71] Shrestha, N. ; Kubler, S. ; Främling, K.: Standardized Framework for Integrating Domain-Specific Applications into the IoT. In: *2014 International Conference on Future Internet of Things and Cloud*, Aug 2014, S. 124–131

[72] Smith, Peter: *Report: Valve OpenVR joins OSVR (open source virtual reality) initiative*. 2015. – URL https://www.itworld.com/article/2923912/personal-technology/report-valve-openvr-joins-osvr-open-source-virtual-reality-initiative.html. – Zugriffsdatum: 2019-08-08

[73] Speicher, Marco ; Ehrlich, Jan ; Gentile, Vito ; Degraen, Donald ; Sorce, Salvatore ; Krüger, Antonio: Pseudo-haptic Controls for Mid-air Finger-based Menu Interaction. In: *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. New York, NY, USA : ACM, 2019 (CHI EA '19), S. LBW0285:1–LBW0285:6. – URL http://doi.acm.org/10.1145/3290607.3312927. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-4503-5971-9

[74] Stauffert, Jan-Philipp ; Niebling, Florian ; Latoschik, Marc E.: Towards Comparable Evaluation Methods and Measures for Timing Behavior of Virtual Reality Systems. In: *Proceedings of the 22Nd ACM Conference on Virtual Reality Software and Technology*. New York, NY, USA : ACM, 2016 (VRST '16), S. 47–50. – URL http://doi.acm.org/10.1145/2993369.2993402. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-4503-4491-3

[75] Steinicke, Frank ; Bruder, Gerd ; Ropinski, Timo ; Hinrichs, Klaus H.: The Holodeck Construction Manual. In: *Proceedings of the ACM International Conference and Exhibition on Computer Graphics and Interactive Techniques (SIGGRAPH) (Conference DVD)*, URL http://basilic.informatik.uni-hamburg.de/Publications/2008/SBRH08. – Zugriffsdatum: 2019-08-08, 2008

[76] Sutherland, Ivan E.: The Ultimate Display. In: *Proceedings of the IFIP Congress*, 1965, S. 506–508

[77] Technologies, Unity: *Unity IDE (official website)*. 2017. – URL https://unity3d.com. – Zugriffsdatum: 2019-08-08

[78] Thielman, Sam: *HoloLens: the virtual reality headset for elevator repair.* 2016. – URL https://www.theguardian.com/business/2016/sep/19/hololens-the-virtual-reality-headset-for-elevator-repair. – Zugriffsdatum: 2019-08-08

[79] Tsai, Yu-Ju ; Wang, Yu-Xiang ; Ouhyoung, Ming: Affordable System for Measuring Motion-to-photon Latency of Virtual Reality in Mobile Devices. In: *SIGGRAPH Asia 2017 Posters*. New York, NY, USA : ACM, 2017 (SA '17), S. 30:1–30:2. – URL http://doi.acm.org/10.1145/3145690.3145727. – Zugriffsdatum: 2019-08-08. – ISBN 978-1-4503-5405-9

[80] United States Department of Justice: *U.S. V. Microsoft: Proposed Findings Of Fact.* 2019. – URL https://www.justice.gov/atr/us-v-microsoft-proposed-findings-fact-0. – Zugriffsdatum: 2019-08-08

[81] U.S. National Institute of Standards and Technology: *Cyber-Physical Systems.* 2019. – URL https://www.nist.gov/el/cyber-physical-systems. – Zugriffsdatum: 2019-08-08

[82] Valve: *OpenVR/SteamVR (github).* 2018. – URL https://github.com/ValveSoftware/openvr. – Zugriffsdatum: 2019-08-08

[83] Van Hee, Lente ; Van Den Heuvel, Ruben ; Verheyden, Tim ; Baert, Denny: *Google employees are eavesdropping, even in your living room, VRT NWS has discovered.* 2019. – URL https://www.vrt.be/vrtnws/en/2019/07/10/google-employees-are-eavesdropping-even-in-flemish-living-rooms/. – Zugriffsdatum: 2019-08-08

[84] Vuforia: *Image Target Guide.* 2019. – URL https://library.vuforia.com/articles/Training/Image-Target-Guide. – Zugriffsdatum: 2019-08-08

[85] Witmer, Bob G. ; Singer, Michael J.: Measuring Presence in Virtual Environments: A Presence Questionnaire. In: *Presence: Teleoper. Virtual Environ.* 7 (1998), Juni, Nr. 3, S. 225–240. – URL http://dx.doi.org/10.1162/105474698565686. – Zugriffsdatum: 2019-08-08. – ISSN 1054-7460

[86] WolframAlpha LLC: *WolframAlpha.* 2019. – URL https://www.wolframalpha.com/. – Zugriffsdatum: 2019-08-08

# A Appendix

## A.1 Experiments

The Omniscope framework and VibeGlove project were implemented by the author. While the hardware and software of the Shelldon prototypes were developed by the author, the prototypes' behavior and carapace were designed in cooperation with Jessica Broscheit. The SpaceFlight and CSTI Opening ceremony simulations were a collaborative effort with Johann Bronsch and other members of the CSTI.

### A.1.1 Omniscope

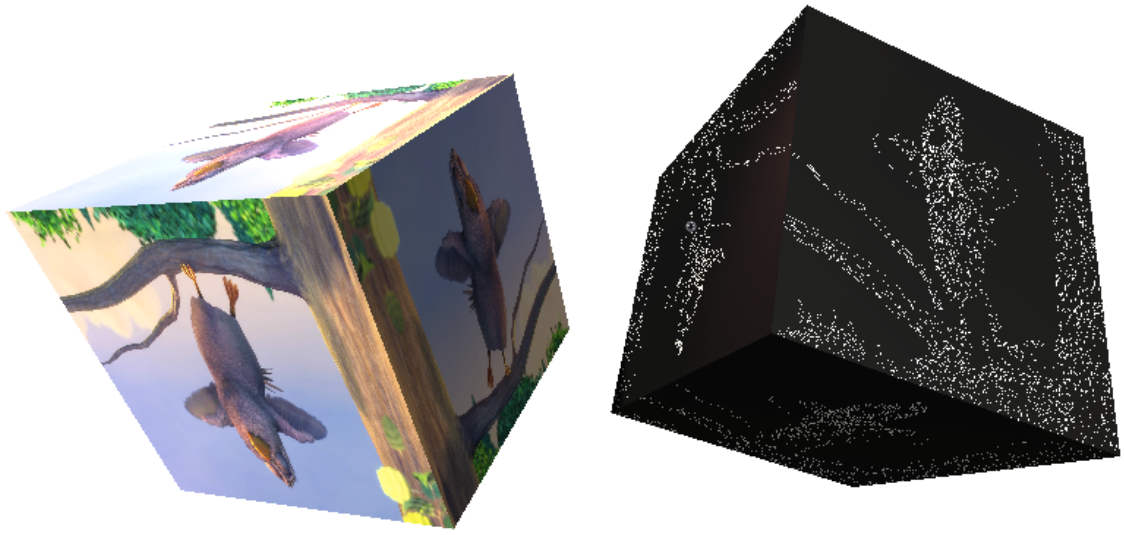


Fig. A.1: Omniscope Plugin rendering and transforming a Full-HD video using edge detection [46]

The Omniscope is a multimedia streaming and computer vision framework. Its' purpose and architecture were already subject of a separate paper ([46]), so this section will purposely be kept short and introductory.

### Motivation

The Omniscope Framework's creation resulted from the lack of an open, freely customizable multimedia streaming integration into the Unity3D IDE used so ubiquitously in the CSTI. The IDE itself offered very minimal playback support for streaming media, limited to only a handful of formats and teeming with bugs that made it very uncomfortable to use. More advanced features and even computer vision was possible with plugins, but these were generally closed-source, proprietary and had other limits of their own.
Thus, it was decided that a new multimedia streaming plugin was to be created for playback of streaming media within Unity3D applications.
In time, it became apparent that another common use case was the *analysis* and *transformation* of multimedia streams, particularly for video. Examples for this include the tracking of image targets using camera arrays and facial recognition. This led to the extension of the Omniscope Project using the OpenCV computer vision library and extended use of GStreamer's own transformation elements.

### Architecture

The Omniscope Project features a highly modular architecture. It is written in C/C++, with another higher level abstraction for the Unity3D IDE written in C#. It has been designed to be usable as a plugin for the Unity3D IDE as well as a standalone binary. This makes it possible to debug features quickly without having to run underneath the IDE's heavyweight processes. It also makes the Omniscope a good fit for some standalone applications like purpose-built video players.
The Framework is highly extensible, featuring an object-oriented pipeline approach modeled after the one used in GStreamer. It offers an intuitive GUI as well as a comparatively high-level C# API and lower level C-style linkages.
Plans exist for a re-implementation of the Omniscope system in the Rust proramming language, which might make the development of further plugins more approachable for beginners.

### A.1.2 SpaceFlight

SpaceFlight was a virtuality application intended to showcase interactions and highlight challenges. It had been developed as part of the CSTI's ongoing research efforts into interaction in the virtuality continuum (see [6]).
SpaceFlight was a virtual reality simulation featuring a virtual spaceship cockpit of approximately $25m^2$ that a (single) user could walk around in freely. The spaceship was situated in a scale-model of the solar system, initially orbiting earth. The cockpit contained a number of objects which the user could interact with, such as freely floating colorful balls that could be pushed around and a large potted plant which changed color when touched by the user or one of the floating balls. It also contained a flight stick which could be used to steer the spaceship. The flight stick existed in two forms: A real flight stick mounted on a pedestal and a digital, three-dimensional reproduction of it in the simulation. The virtual reproduction was carefully arranged to superimpose over the real flight stick.
The simulation was intended primarily to showcase a number of forms of *manual* interaction: The primary mode of interaction was the user's own hands as tracked by the LeapMotion system. The user could see virtual representations of their own hands in near real-time. They could use their hands to push the floating balls around or trigger buttons. This was made possible using the simulation's collision detection and physics features on the hand representations controlled by the LeapMoption system.
The real flight stick allowed the user to control the spaceship's speed as well as rotation around all three axes. The flight stick's own movement and rotation was extrapolated from sensor data received from it and reflected in its virtual representation. This allowed the user to seemingly grasp and manipulate the virtual representation physically.
Lessons learned from the experiment included some of the limits in current hand tracking technology, such as difficulties recognizing exact finger placement when grasping (a physical object or even a virtual one). It did however also show that users were able to interact with purely virtual objects with a very high degree of precision using their hands. even in the absence of haptic feedback.

### A.1.3 VibeGlove

The VibeGlove was a strictly minimalist implementation of a tactile feedback system in glove-shape, triggered by colliders in the simulation. Two designs were created, one

wired directly to the simulation's host computer via a USB serial connection, the other using a wireless LAN connection.

### Hardware

The VibeGlove consisted of three sections: A microcontroller, a power supply (battery) and an array of five vibration motors with driver circuits.
The microcontroller used differed in two iterations. The first was an AVR microcontroller chosen for its ease of integration with the CSTI's development infrastructure. The board it was seated on which contained all but the vibration motor and battery circuits was similar to an Arduino Nano. The primary drawback of this design was the reliance on direct, wired serial connections (USB). In its second iteration, the VibeGlove featured an ESP32-based microcontroller board, adding connectivity via Bluetooth and Wireless LAN. Since that board also featured built-in low-dropout voltage regulators, the integration of a battery was easier than in the first design.
The vibration motors were simple, 60mA coin-cell size vibration motors of a type commonly found in cellphones.
The VibeGlove was affixed to a user's hand by means of a skeletal structure, leaving as much skin exposed as possible so as not to interfere with camera based motion tracking (the LeapMotion system uses stereoscopic infrared cameras, see [70] for a teardown of the system and analysis of its technical composition).

### Software

The VibeGlove's software consisted of two distinct parts: A firmware running on the microcontroller and a client/driver software for the VC simulation's host computer.
The firmware was written in the Arduino C/C++ dialect. The switch from an AVR to ESP32 microcontroller prompted a partial rewrite of the software. The client/driver software was written in C#. Both Arduino and C# were chosen for their wide adoption and comparatively shallow learning curve when compared to the alternatives (C/C++). C# also enjoys extensive support from within the Unity IDE.

### A.1.4 CSTI Opening Ceremony

The CSTI Laboratory's Opening Ceremony Application was a VR application which was showcased during the official opening of the laboratory. The simulation had the following goals:
Firstly, it showcased two experimental gesture interaction scenarios. One was the cutting of the rope using one's hand (with a visual feedback channel, i.e.: the user could see their hand). The other was the option to shove wall fragments (also using one's hand), in a simulated low gravity environment.
Secondly, it served as an experiment to gain insights into the effect of sudden expansion of the perceived space surrounding the user.
Thirdly, the opening ceremony application was developed under relatively tight time constraints, which made its economic, goal oriented and minimalist approach paramount to its success. As such, it serves as an example of an economical VC application.

#### Hardware

The CSTI Opening Ceremony system consisted of the simulation's host computer, an HTC Vive head-mounted display and a LeapMotion hand tracking camera system (affixed to the HMD). It took place in the CSTI's $25m^2$ motion-tracked VR space.

#### Software

On the software side, the CSTI Opening Ceremony simulation was composed using the Unity3D IDE. Programming was done exclusively in C# using Microsoft Visual Studio. Assets and models were made primarily using Blender.

#### Presentation and Behaviour

The CSTI Opening Ceremony Application was intended to act as a showcase at the CSTI's eponymous event. The simulation showed the user a digital representation of the CSTI itself, with them standing in the same exact spot as in reality. One wall of the $25m^2$ motion-tracked area of the lab was displayed as a brick wall. In front of this wall hung a bottle from two ropes in mid-air. In a metaphorical reference to a ship's

launching ceremony,[1], the user could use their hand (a digital representation based on real-time motion tracking could be seen) to cut the holding rope, causing the bottle to swing towards the wall. Instead of smashing itself, however, the bottle would break the wall into fragments, opening the wall to a simulation of outer space, with the laboratory seemingly in a high orbit above earth, with a crystalline-looking logo of the CSTI spinning above it.
It was the intention of this simulation to act in a dual purpose: Firstly, as a showcase of virtual reality technologies to the assembled spectators of the CSTI's opening ceremony. Secondly, as an experiment to observe the impact of a sudden break in realism from a relatively realistic digital representation of the user's actual surroundings to a completely impossible infinite space scenario.
During the presentation, an audience could see a projection of the user's perspective projected onto a canvas in front of them. Subsequent users therefore already had an idea of what to expect from the situation.

**Observations and lessons learned**

Users quickly realized they could see their hands inside the VR simulation and interact with objects by touching them. That touching the rope would cause it to be severed, however, needed explaining - it was not intuitive.
Even when they knew what to expect (as they had been watching prior uses of the simulation through the ego-perspective projection), users still acted surprised and tense when the confined space around them seemed to open up to space. A particular fascination seemed to stem from the feeling of height above the earth; they instinctively avoided standing too close to the edge of the room.
In terms of its construction, the simulation was a satisfactory success despite the narrow development time constraints. The creation of a virtual representation of the real laboratory environment took a considerable part of this time. In the end, this was somewhat mitigated through the use of dim lighting and ambient occlusion.
The most challenging programming aspects of the simulation were related to the interactions offered. Manual interaction using the user's motion tracked hands were not especially difficult to implement since both the motion tracking and HMD systems manufacturers offered integrations for the Unity IDE. Customizing these to better fit the

[1] which itself was intended as a nod to Hamburg's nature as a major sea port and member of the hanseatic league

purpose was considerably harder, especially where the simulation of realistic physics came in. Eventually, the development team settled for a constant low-gravity setting where the movement of the ropes, bottle and wall fragments were intentionally slowed down, both for dramatic and easily observable effect and so as not to have an easily observable change in physics between normal gravity 'room' and near-zero-gravity 'space' settings.

### A.1.5 Shelldon Project

The Shelldon IoT/Smart Object project is another showcase for the CSTI laboratory. It consists of a five smart objects and a host station. The objects resemble a bale of semi-translucent, plastic garden turtles. Using an array of LEDs, they can glow in a variety of complex light patterns. They are also capable of sensing how they are held and of communicating with one another, their host station and the world.
The individual, physical Shelldon prototypes are differentiated by means of Greek characters (in order to avoid confusion between the physical prototypes and their development iterations which are named using the Latin alphabet).

#### Hardware

The prototypes' shells were 3D-printed out of semi-translucent sparkling ABS plastic by means of fused filament deposition[2]. The model used - offered through Thingiverse by user 'pmoews' under a Creative Commons Attribution license, see [64] - is allegedly based on a 3D scan of a garden turtle. It was slightly altered, resized and hollowed out to accomodate the electronic components.
Inside their shells, the Shelldons' interior consists of the following:

- an Adafruit Feather HUZZAH board (essentially a breakout board with an Espressif ESP8266 microchip)
- an Adafruit NeoPixel FeatherWing 4x8 LED matrix shield
- a power supply circuit incorporating a compatible 800mA 3.7V LiPo battery with an on/off toggle switch (mounted into a plastic bottom plate which can alternatively be removed)

[2] using an 'Ultimaker 3 Extended' 3D printer

The Adafruit Feather HUZZAH board and NeoPixel FeatherWing shield were soldered together using flexible wire connectors rather than standard pin headers in order to minimize the height of the gap between them and fit into their cases.
The host station consists of a Raspberry Pi 3 B+ with a simple case (also 3D-printed out of PLA). It is powered using a 5V rechargeable power bank based on standard rechargeable 18650 Li-Ion batteries.
The Shelldon prototypes and the host station are kept in a protective foam-filled silver aluminum presentation case.

Fig. A.2: 'Shelldon' prototype (resting mode)

**Software**

The Shelldon prototypes themselves were programmed in the Arduino programming language. This was a conscious design decision to make further development and maintenance as approachable as possible for beginners.
The Host station runs an Arch Linux ARM OS with a minimalist software stack of

NodeRED ([58]), Festival ([10]) and the Mosquitto MQTT broker ([18]). NodeRED is used to define the Shelldon's behaviour in a simple manner (and with graphical representation for many aspects). MQTT is a message queueing system for machine-to-machine communication (see ch. 5).

### Presentation and Behaviour

The Shelldon prototypes are designed to showcase and highlight some of the potential of Smart Devices, particularly with regards to machine-to-machine and human-machine communication: They communicate with one another and their host station using MQTT, their users using their rotational sensors as input and LED matrices as output as well as with their 'owner' by means of the NodeRED web interface. In addition, they are capable of calling an emergency phone number, using the Festival speech synthesis system ([10]) to talk, as well as of sending (and receiving) messages via twitter. When presented to an audience - e.g. as part of a slideshow presentation - they glow calmly in shifting patterns of blue and green. If they are held at a significant angle (of more than 30 degrees off the norm any axis), their glow shifts to orange hues to indicate a warning. When turned on their side or back, they pulsate in a very intense red. This indication of 'distress' is mirrored on all Shelldon prototypes, representing a state of empathetic stress in the bale. Only when all turtles are turned on their belly again will they stop panicking and return to their natural hues.
In the state of extreme distress, they are also capable of sending out a distress call via Twitter and calling a predefined emergency number (typically a phone on the presenter's desk or a cellphone in the audience). In both instances, the distressed individual indicates its distress, its identity and its desire to be turned back on their belly, e.g.: "This is Shelldon Alpha! I have been turned on my back and can't get up! Please put me back on my belly!". These advanced communication abilities were presented during workshops in the laboratory but are turned off by default so as not to require an internet connection when they are presented elsewhere. Instead, the host station acts as an independent wireless local area network access point for the Shelldon prototypes.

## A.2 Devices and Technologies

### A.2.1 Electrophoretic Displays

These displays do not emit light, instead consisting of a matrix of cells whose color (i.e. reflective nature) can be changed by applying precisely modulated electrical impulses. These displays have a number of benefits over traditional light-emitting displays: Their contrast characteristics when subjected to another light source increase rather than decreasing and they only consume power upon changing the shown image. Drawbacks include a typically very low refresh rate (up to several seconds) and limitation of the displayable color spectrum: Most EPDs are limited to only displaying black and white, though some exist that can also display red, yellow or shades of gray.
EPDs are obviously unsuited for use in HMDs, but they offer interesting applications in other niches. Most prominently, they are being used to simulate the look of printed and even handwritten pages of paper, which could be considered an Augmented Reality application of sorts. Their extremely low power consumption and readability under direct (sun)light also makes them ideal for Wearables and Smart Objects which need to communicate visually with the user.

# B Glossary

**Virtuality** The term virtuality is intended to encompass all systems and experiences of the virtuality continuum (cf. ch. 1.1) and therefore largely synonymous with Virtuality Continuum (or VC). Cf. ch. 1.

**Virtuality System** The combination of technical (hardware), programmatic (software) and human aspects in order to achieve a virtuality experience (cf. virtuality above).

**Head-Mounted Display (HMD)** Visual display system designed to be worn on the user's head. Typically stereoscopic, though single-eye variants exist. Cf. ch. 2.

**Polymorphic Interaction Scenarios (PIS)** A proposed means of generically describing interactions in the VC by their intent and the arguments required. Cf. ch. 2.

**Access Conditions** First part of the proposed set of requirements categories for the integration of interaction objects. Cf. ch. 3.

**Communication Requirements** Second part of the proposed set of requirements categories for the integration of interaction objects. Cf. ch. 3.

**Interaction Requirements** Third part of the proposed set of requirements categories for the integration of interaction objects. Cf. ch. 3.

## Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

*Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI*

## Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Masterarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

**Towards Distributed Interaction Systems in the Virtuality Continuum**

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

| _____ | _____ | _____ |
|---|---|---|
| Ort | Datum | Unterschrift im Original |