Universität des Saarlandes
Fakultät für Mathematik und Informatik (MI)
Fachrichtung Medieninformatik

A Dual Reality Approach for Spatial Inference, Planning, Prediction and Prototyping

Bachelorarbeit

vorgelegt von

Andreas Dieter Fey

am 09. Januar 2017

Angefertigt und betreut unter der Leitung von
Christian Bürckert

Begutachtet von
Prof. Dr. rer. nat. Dr. h.c. mult. Wolfgang Wahlster
Dr. Tim Schwartz

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, ...

(Datum / Date) (Unterschrift/ Signature)

Kurzzusammenfassung

Die Zusammenarbeit von Mensch und Maschine wird als Mensch-Roboter Kollaboration (MRK) bezeichnet. In MRK-Szenarien muss der Roboter zu jedem Zeitpunkt wissen wo er selbst und wo der Mensch ist, da der Roboter sonst zu einer Gefahr für den Menschen werden kann. Je mehr Menschen und Roboter Teile eines Szenarios sind, desto wichtiger wird es, dass die Roboter genau wissen, wo die Menschen sind. Wenn die Roboter dies nicht wissen, sind sie eine Gefahr für die Menschen die am Szenario beteiligt sind. Da ein Roboter alleine meist nicht in der Lage ist, eigenständig auf alle Teilnehmer des Szenarios aufzupassen, wird in der Regel eine virtuelle Repräsentation der Umgebung zur Verfügung gestellt, über die der Roboter sich orientieren kann. Dieses Zusammenbringen der echten und der virtuellen Welt ist keine triviale Aufgabe, da die Sicherheit der Menschen in solch einer Zusammenarbeit zu jedem Zeitpunkt sichergestellt sein muss. Aus diesem Grund wird meistens schon viel Zeit in das Grundgerüst solch eines MRK-Szenarios investiert, damit auch alles, was auf dem Gerüst aufbaut, die richtigen und notwendigen Information abrufen kann. Um das Erstellen eines MRK-Szenarios zu beschleunigen, kann der TEDURU-Server benutzt werden, welcher für diese Arbeit entwickelt wurde und als Rahmenwerk für MRK-Szenarien benutzt werden kann. Um möglichst vielseitig einsetzbar zu sein, baut der TEDURU-Server auf Methodiken der Dual Reality auf.

Inhaltsverzeichnis

1 Einführung **1**
- 1.1 Motivation . . . 3
- 1.2 Zielsetzung . . . 4
- 1.3 Gliederung . . . 5

2 Verwandte Arbeiten **6**
- 2.1 Auswahl genutzter Rahmenwerke . . . 6
  - 2.1.1 Second Life . . . 6
  - 2.1.2 OpenSimulator . . . 9
  - 2.1.3 C# Microsoft .Net . . . 13
  - 2.1.4 Java 3D . . . 14
- 2.2 Verwandte Systeme . . . 15
  - 2.2.1 Darstellung von Kontextfeldern (Lifton 2007) . . . 15
  - 2.2.2 Twin World Mediator (Brandherm 2008) . . . 17
  - 2.2.3 Eolus One (Coleman 2009) . . . 19
  - 2.2.4 Virtuelle Schokoladenfabrik (Back et. al 2010) . . . 19
  - 2.2.5 Yamamoto (Stahl und Haupert 2006, Stahl 2009) . . . 21
  - 2.2.6 DuRMaD (Kahl 2014) . . . 23
- 2.3 Zusammenfassung . . . 25
  - 2.3.1 Zugrundeliegende Systeme im Vergleich . . . 26
  - 2.3.2 Verwandte Systeme im Vergleich . . . 29

3 Konzept und Implementierung **32**
- 3.1 Kommunikation und Rahmenwerk . . . 32
  - 3.1.1 Konzept . . . 32
  - 3.1.2 Implementierung . . . 32
- 3.2 Visuelles Debugging . . . 33
  - 3.2.1 Konzept . . . 33
  - 3.2.2 Implementierung . . . 34
- 3.3 Automatische Integration von Modellen . . . 35
  - 3.3.1 Konzept . . . 35
  - 3.3.2 Implementierung . . . 36
- 3.4 Laden von Modellen . . . 39
  - 3.4.1 Konzept . . . 39
  - 3.4.2 Implementierung . . . 39
- 3.5 Transformationen . . . 39
  - 3.5.1 Konzept . . . 39
  - 3.5.2 Implementierung . . . 40
  - 3.5.3 Position . . . 41
  - 3.5.4 Rotation . . . 41
  - 3.5.5 Skalierung . . . 41
- 3.6 Speichern und Laden von Szenen . . . 41
  - 3.6.1 Konzept . . . 42

3.6.2 Implementierung . . . 42
3.7 Kollisionen Erkennen . . . 43
3.7.1 Konzept . . . 44
3.7.2 Implementierung . . . 44
3.8 Raycasting . . . 46
3.8.1 Konzept . . . 46
3.8.2 Implementierung . . . 46
3.9 Ankerpunkte . . . 47
3.9.1 Konzept . . . 47
3.9.2 Implementierung . . . 47
3.10 Hexagonkarten . . . 48
3.10.1 Konzept . . . 48
3.10.2 Implementierung . . . 48
3.11 Zusätzliche Informationen zu Objekten . . . 49
3.11.1 Konzept . . . 49
3.11.2 Implementierung . . . 49
3.12 Zurverfügungstellen von Informationen . . . 50
3.12.1 Konzept . . . 50
3.12.2 Implementierung . . . 50
3.13 Benutzerinterface im Server . . . 51
3.13.1 Konzept . . . 51
3.13.2 Implementierung . . . 51

4 Evaluation **53**
4.1 Einführung . . . 53
4.2 Motivation . . . 53
4.3 Zielsetzung . . . 54
4.4 Konzept . . . 54
4.5 Implementierung . . . 54
4.5.1 Hololens Implementierung . . . 56
4.5.2 Marvelmind + Smartwatch Implementierung . . . 57
4.6 Ergebnis . . . 58

5 Konklusion **59**

6 Weiterführende Arbeiten **60**

Literatur **62**

1 Einführung

Der Mensch strebt schon seit Urzeiten nach Mitteln und Wegen Arbeiten und Aufgaben zu vereinfachen, oder gar zu automatisieren. Angefangen hat es mit Werkzeugen, welche definiert werden als „*nicht zum Körper gehörendes Objekt, mit dessen Hilfe die Funktionen des eigenen Körpers erweitert werden, um auf diese Weise ein unmittelbares Ziel zu erreichen*“ [52]. In Abbildung 1a sind einige solcher modernen Werkzeuge abgebildet. Mit steigender Komplexität der Aufgaben sowie einer stetigen Erhöhung der hergestellten Stückzahl, haben Werkzeuge nicht mehr zur nötigen Entlastung des Arbeiters ausgereicht. Dies führte zur Entwicklung von Maschinen. So ist es nicht sinnvoll zu versuchen ein Frachtschiff mit Ruderern zu bewegen, stattdessen besitzen Frachtschiffe Maschinen wie in Abbildung 3a, welche die Schiffsschraube drehen und somit das Frachtschiff, fortbewegen. Maschinen sind meist nur für eine Aufgabe konzipiert und haben keinerlei brauchbare Funktionalität außerhalb dieser Aufgabenstellung. Aus diesem Grund hat der Mensch seine Maschinen zu Robotern fortentwickelt. Roboter hingegen sind aufgrund ihres Aufbaus und der Tatsache, dass sie programmierbar sind, meist für viele verschiedene Aufgaben nutzbar. In Abbildung 3b ist solch ein programmierbarer Mehrzweckroboter abgebildet. Für lange Zeit waren Roboter und Maschinen von menschlichen Arbeitern getrennt. Wenn ein Mensch einem Roboter zu nahe gekommen ist, so musste der Roboter immer, zur Sicherheit des Menschen, ausgeschaltet werden. Der nächste große Schritt in der Entwicklung ist eine direkte Zusammenarbeit von Mensch und Roboter in der gleichen Umgebung, ohne trennende Schutzeinrichtungen. Solch ein Szenario wird auch als Mensch-Roboter Kollaboration (MRK) oder entsprechend im englischen als *Human Robot Collaboration* (HRC) bezeichnet [1].

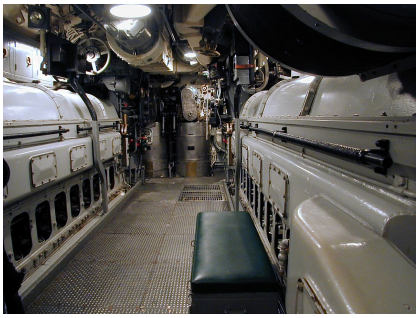
Vorteil von MRK-Szenarien ist, dass diese im Gegensatz zu normalen Roboteranwendungen nicht an Fabrik-, Werkstatt- oder Laborumgebungen gebunden sind. Durch MRK sind Roboter in der Lage das Leben von Menschen sicherer und einfacher zu gestalten. Beispiele hierfür sind Roboter in der Altenpflege [41], in der Raumfahrt [5] oder auch in Rettungseinsätzen [38].

Es haben sich jedoch nicht nur Maschinen und Roboter weiterentwickelt. In Einklang mit *ubiquitous computing* (Ubiquitäres Computing) [54] sind Computer und Sensoren immer kleiner und effektiver geworden. Ubiquitäres Computing besagt, dass Rechengeräte immer allgegenwärtiger werden und nach der Vision von Mark Weiser auch irgendwann persönliche Computer als einzelne Geräte ersetzen. Somit ist es jetzt durchaus denkbar, Computer und Sensoren überall anzubringen. Beispiele hierfür sind sogenannte *wearables* (tragbare Computersysteme). Dabei ist der Unterschied zwischen *wearable computing* zu normalen tragbaren Computersystemen, dass der Nutzer bei dem Tragen eines *wearables* dieses unterstützend zu seiner Tätigkeit bei sich trägt, es jedoch nicht die primäre Tätigkeit des Nutzers ist, mit dem *wearable* zu interagieren. Beispiele sind intelligente Textilien [42, 35], *Smartwatches* oder auch die Microsoft Hololens [2].

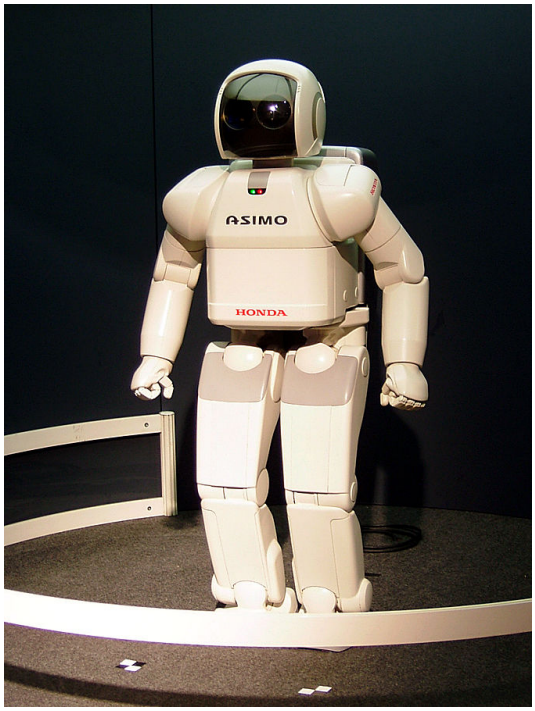
Sensoren können auch eingesetzt werden um Systeme zu überwachen. Hierbei wird von *Cyber-Physical Systems* (CPS) gesprochen. Der Begriff der CPS wurde schon früh von Edward Lee geprägt und wie folgt definiert: „*Cyber-Physical Systems (CPS) are integrations of computation and physical processes. Embedded computers and networks monitor and control the physical processes, usually with feedback loops where physical processes affect computations and vice versa.*“ [32]. Zu deutsch etwa: *Cyber-Physische Systeme sind Einbindungen von Be-*

[1] `https://www.kuka.com/en-us/technologies/human-robot-collaboration`
[2] `https://www.microsoft.com/en-us/hololens`

(a) *Werkzeuge. Quelle:* `https://goo.gl/A7A7Wr`



(b) *Schiffsmaschinen. Quelle:* `https://goo.gl/v11eVS`



(c) *Roboter. Quelle:* `https://goo.gl/KMYnrz`

Abbildung 1: *Werkzeuge, Maschinen und Roboter.*

rechnungen und physischen Prozessen. Eingebettete Computer und Netzwerke überwachen und kontrollieren die physischen Prozesse für gewöhnlich mit Rückkopplungsschleifen, bei denen physische Prozesse und Berechnungen sich gegenseitig beeinflussen. In Abbildung 2 sind die vier industriellen Revolutionen bis hin zur vierten industriellen Revolution, in welcher CPS von Bedeutung sind, skizziert. Gefördert wurden Projekte zu CPS erstmals in den USA im Jahr 2007 als das *President's Council of Advisors on Science and Technology* (PCAST) ein Forschungsprogramm der *National Science Foundation* (NSF) mit dem Titel *Cyber-Physical Systems* eingerichtet hat und diese im März 2008 weitergeführt hat [3]. In Deutschland hat die Deutsche Akademie der Technickwissenschaften (acatech) [4] in Zusammenarbeit mit dem Bundesministerium für Bildung und Forschung (BMBF) [5] eine *Forschungsagenda CPS* erarbeitet [21, 9].

Eine Ausweitung eines CPS auf eine Umgebung wird als *Cyber-Physical Environment* (CPE) bezeichnet [39]. CPE's erweitern natürliche Umgebungen wie Büros, Fabriken oder auch Privathaushalte, wie zum Beispiel von Amazons *Smart Home* Produkten *Alexa* und *Echo* demonstriert wird [6]. Solch eine CPE kann in einem Privathaushalt durch den Kauf von entsprechenden Produkten nach Wunsch und momentanem Verlangen eingekauft und

[3] `https://www.ece.cmu.edu/~electriconf/2008/PDFs/Gill%20-CMU%20Electrical%20Power%202008%20-%20%20Cyber-Physical%20Systems%20-%20A%20Progress%20Report.pdf`
[4] `http://acatech.de`
[5] `https://www.bmbf.de`
[6] `https://amazon-presse.de/Kindle---Fire/Echo-und-Alexa/Newsdetail/amazon/de/Devices/2016-09-14-Echo---Dot/`

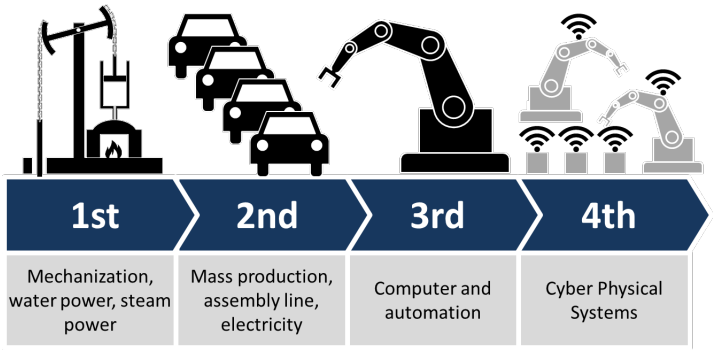


Abbildung 2: *Die vier Industriellen Revolutionen hin zu CPS. Quelle:* `https://goo.gl/qyGWMM`

installiert werden. In einem Bürokomplex oder auf einem Fabrikgelände hingegen muss die Integration einer CPE ausführlich geplant und geprüft werden, damit sie den erforderlichen Anforderungen entspricht.

In Abhängigkeit der in einer CPE installierten Sensoren kann diese auch um eine *Dual Reality* Anwendung erweitert werden. Das Konzept der Dual Reality wurde erstmalig von Lifton als *An environment resulting from the interplay between the real world and the virtual world, as mediated by networks of sensors and actuators. While both worlds are complete unto themselves, they are also enriched by their ability to mutually reflect, influence, and merge into one another.* definiert [34].

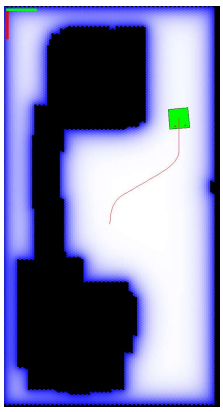
Dual Reality Anwendungen eignen sich in Kombination mit einer CPE, um MRK-Szenarien zu unterstützen. Die CPE liefert Informationen über Positionen und Zustand der Aktuatoren in dieser Umgebung. Die Aktuatoren umfassen hierbei sowohl menschliche Teilnehmer, als auch Roboter. Somit können die Aktuatoren über die Dual Reality Anwendung mit der Umgebung kommunizieren, wodurch eine Umsetzung des gewünschten MRK-Szenarios vereinfacht wird.

Mit Hilfe des in dieser Arbeit entwickelten Dual Reality Systems, wird die Erstellung von MRK-Szenarien vereinfacht. Zudem wird die Instandhaltung der MRK-Szenarien vereinfacht, was sich ebenfalls positiv auf deren Robustheit auswirken kann.

1.1 Motivation

In *Human Robot Collaboration: A Survey* [11] wurde schon angedeutet, dass in MRK-Szenarien eine Vielzahl von Forschungsfeldern existieren, die selbst für interdisziplinäre Forschungsgruppen große Herausforderungen bilden. Als Beispiel für solche Herausforderungen werden Verhalten, Aussehen sowie kognitive und soziale Fähigkeiten von Robotern genannt. So braucht es interdisziplinäre Kooperationen zwischen Psychologen, Verhaltensforschern und klassischen Roboterforschern um sich diesen Forschungsthemen zu stellen. Dementsprechend sollten solche Forschungsgruppen nach Möglichkeit in der Lage sein, sich nur um die Forschung auf ihrem Fachgebiet zu kümmern, ohne zu viel Zeit in die grundlegende Architektur des Forschungsszenarios stecken zu müssen.

Das Erstellen einer Architektur für ein MRK-Szenario ist kein triviales Problem und kann viel Zeit in Anspruch nehmen, die vom Grundsatz her für andere Aufgaben besser genutzt



(a) *Digital geplanter Pfad*

(b) *Baxter Roboter während der Ausführung des Plans*

Abbildung 3: *Wegfindung eines Roboters über eine digitale Karte der Umgebung*

werden kann. Ein Problem, welches von einer Architektur für MRK-Szenarien gelöst werden muss, ist unter anderem, die Synchronisation von Aktuatoren und Agenten der echten Welt mit der virtuellen Welt. Die virtuelle Welt dient nicht-menschlichen Komponenten eines MRK-Szenarios meist als Grundlage, um verschiedenste Aufgaben zu lösen. Eine typische Aufgabe für die virtuelle Repräsentation einer realen Umgebung ist die Wegplanung von mobilen Roboterplattformen in dieser Umgebung [26, 31, 17], wie auch in Abbildung 3 zu sehen ist. Probleme, die von einer Architektur für MRK-Szenarien behandelt werden müssen, umfassen verschiedene Koordinatensysteme von Agenten und Aktuatoren des Szenarios, das Einfügen von virtuellen Modellen sowie das Zurverfügungstellen von nötiger Funktionalität für die einzelnen Modelle oder auch, dass die verwendete Kommunikationsinfrastruktur von allen Komponenten des Szenarios verwendet werden kann. All diese Probleme sind meist nicht forschungsrelevant und sollten nicht für jede Umsetzung eines MRK-Szenarios neu implementiert werden müssen.

In dieser Arbeit soll daher ein *Framework* (zu deutsch Rahmenwerk) für MRK-Szenarien entwickelt werden, welches das Erstellen und Ausführen dieser vereinfachen soll. Um das Erstellen und Ausführen von MRK-Szenarien zu ermöglichen nutzt diese Arbeit Dual Reality Methoden, um die echte Welt mit der virtuellen Welt zu synchronisieren.

1.2 Zielsetzung

Im Kontext dieser Arbeit stehen zwei Ziele im Mittelpunkt, die beim Erstellen von MRK-Szenarien zu erfüllen sind. Diese beiden Ziele werden im Folgenden aufgelistet und erklärt.

- **Die Erschaffung eines Rahmenwerkes zur Vereinfachung und Beschleunigung der Erstellung von Prototypen und fertigen Produkten in MRK-Szenarien.**

  Bei der Umsetzung eines MRK-Szenarios ist es notwendig, dass alle Aktuatoren des Szenarios auf dem gleichen, aktuellen Wissensstand über den Zustand ihrer Umgebung sind, andernfalls können die Roboter des Szenarios für menschliche Teilnehmer des Szenarios eine Gefahr darstellen. Hierfür kann ein zentrales Weltmodell erstellt werden, über das sich die einzelnen Aktuatoren synchronisieren. Die Erstellung solch eines synchronisierten Weltbildes ist jedoch zeitaufwändig und dementsprechend auch fehleranfällig, da die

Synchronisation einer realen mit einer virtuellen Umgebung nicht trivial ist. Deshalb ist es das Kernziel dieser Arbeit ein Rahmenwerk zu erstellen, welches es vereinfachen soll, die echte Welt mit der virtuellen Welt zu synchronisieren.

- **Die Definition eines wiederverwendbaren Kommunikationsprotokolls für MRK-Szenarien.**

  Um der Kernanforderung der Arbeit gerecht zu werden, über ein möglichst vielseitig nutzbares Kommunikationsprotokoll zu verfügen, soll das Rahmenwerk nicht auf einem Kommunikationsprotokoll basieren, welches für ein spezialisiertes MRK-Szenario optimiert wurde. Somit soll das fertige Produkt für möglichst viele verschiedene MRK-Szenarien wiederverwendet werden können.

Abgrenzungen

Das Ergebnis dieser Arbeit soll ausschließlich ergänzend zu MRK-Szenarien benutzt werden, um das Erstellen dieser zu vereinfachen. Ziel dieser Arbeit ist es nicht, einen Baukasten für MRK-Szenarien zu erstellen. Es wird somit nicht möglich sein, ein MRK-Szenario nur mit dem in dieser Arbeit erstellten TEDURU-Server zu implementieren oder zu kontrollieren.

1.3 Gliederung

In Kapitel 2 werden verwandte Techniken und Ansätze vorgestellt, die zur Entscheidungsfindung beigetragen haben. Da schon mehrere Dual Reality Rahmenwerke entwickelt wurden, können deren Methoden, Ansätze und Erfahrungen als Grundlage für diese Arbeit herangezogen werden. Am Ende von Kapitel 2 werden die gesammelten Ergebnisse über die genutzten Rahmenwerke, sowie die Systeme, die mit ihnen implementiert wurden, verglichen. Da ein Rahmenwerk für MRK-Szenarien viele zum Teil auch kürzere, weniger differenziertere oder auch voneinander unabhängige Konzepte implementieren muss, werden in Kapitel 3 die einzelnen Konzepte sowie deren Implementierung nacheinander vorgestellt. In Kapitel 4 wird ein MRK-Szenario vorgestellt, welches mit zwei unterschiedlichen Systemen umgesetzt werden soll. Dieses MRK-Szenario wird mit dem in dieser Arbeit entwickelten TEDURU-Server implementiert, um die Funktionalität des TEDURU-Servers zu überprüfen. Kapitel 5 fasst die Arbeit und die in dieser Arbeit erreichten Ziele zusammen, während in Kapitel 6 Aussichten auf mögliche zukünftige Erweiterungen und Verbesserungen gegeben werden.

2 Verwandte Arbeiten

In diesem Kapitel werden einige Systeme vorgestellt, welche anhand von Sensordaten und Zustandsinformationen aus der echten Welt, Objekte in einer virtuellen Welt beeinflussen und manipulieren. Weiterhin können diese Systeme, über den Zustand ihrer virtuellen Welt, Einfluss auf die physische Welt nehmen. Somit können sich beide Welten gegenseitig beeinflussen und dadurch ergänzen. Hierzu werden im ersten Abschnitt des Kapitels die Rahmenwerke, die von den Dual Reality Systemen benutzt werden, vorgestellt und beschrieben. Im Anschluss werden die Systeme selbst im Detail vorgestellt. Am Ende dieses Kapitels werden Erkenntnisse aus diesen Arbeiten zusammengefasst und mit dem in dieser Arbeit vorgestellten Ansatz, dem TEDURU-Server, verglichen.

2.1 Auswahl genutzter Rahmenwerke

Jedes System basiert auf einem grundlegenden Rahmenwerk. Jedes Rahmenwerk wird mit einem spezifischen Anwendungsfall als Ziel erstellt. Solch ein spezifischer Anwendungsfall kann etwas Konkretes sein, wie das Erstellen von Gif-Animationen aus einzelnen Bildern [7]. Das Ziel eines Rahmenwerkes kann es aber auch sein, ein breites Feld an Funktionen abzudecken, wie zum Beispiel, das Erstellen von Internetseiten [8] [9].

Jede dieser Anwendungen sticht meist durch besondere Merkmale auf dem angestrebten Gebiet hervor, für das es entwickelt wurde. Für Anwendungsfälle außerhalb des optimierten Anwendungsbereiches sind diese Systeme jedoch meist nur mangelhaft ausgestattet. Des Weiteren deckt meist keines der Systeme sein spezielles Anwendungsfeld vollständig ab. Es gibt zumindest immer subjektive Vorlieben der Nutzer, was zu unterschiedlich aufgebauten Benutzerinterfaces und Arbeitsabläufen führt. Somit existieren meist mehrere Rahmenwerke für den gleichen Anwendungsbereich.

Dementsprechend existieren viele Ansätze und Rahmenwerke die benutzt werden können um ein Dual Reality System zu entwickeln. Im Folgenden werden Rahmenwerke vorgestellt, auf denen, die Systeme die im Kapitel 2.2 beschrieben werden, aufbauen. Im Unterkapitel 2.3 werden sowohl die Rahmenwerke, als auch die Systeme selbst, miteinander verglichen.

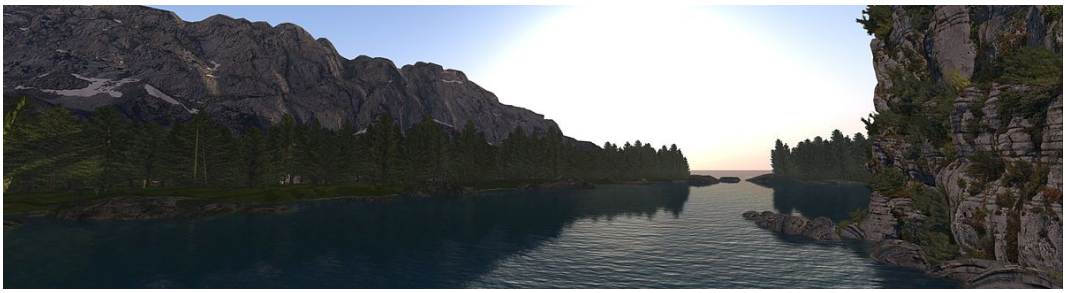
2.1.1 Second Life

Second Life (deutsch: zweites Leben, abgekürzt „SL“, Logo in Abbildung 4a) ist eine von *Linden Lab* [10] entwickelte Software. Linden Lab wurde 1999 von Philipp Rosedale gegründet und hat seinen Hauptsitz in San Francisco. SL wurde am 23. Juni 2003 erstmals veröffentlicht. Die Klassifikation von SL ist nicht eindeutig, so gibt es Ähnlichkeiten zu *Massive Multiplayer Online Role-Playing Games* (MMORPGs), wie zum Beispiel *World of Warcraft* (WoW). SL und MMORPGs verbinden riesige virtuelle Welten und Interaktionen mit diesen virtuellen Welten durch Benutzer, welche selbst durch virtuelle Avatare repräsentiert werden. SL bietet jedoch kein Ziel, kein Ende und auch keinen traditionellen Spielverlauf. Des Weiteren hat Rosedale selbst, im März 2006, in einem *GoogleTecTalks* [11] gesagt, dass SL kein Spiel, sondern eine Plattform ist.

[7] https://gif-erstellen.com/
[8] http://www.nvu.com/
[9] https://www.microsoft.com/expression/eng/index.html
[10] https://www.lindenlab.com/
[11] http://www.allreadable.com/68357nm1

(a) *Second Life logo*
(Quelle: https://goo.gl/AR2im4)

(b) *Mehrere Avatare in SL*
(Quelle: https://goo.gl/UV6oRw)



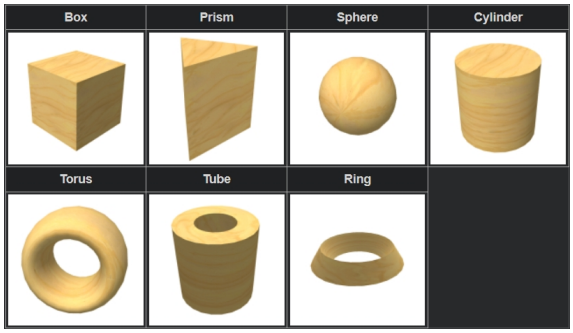
(c) *eine virtuelle Welt in Second Life*
(Quelle: https://goo.gl/cZECWR)

Abbildung 4: *Second Life Logo, sowie Avatare und eine virtuelle Umgebung aus Second Life*

Die Welt von SL besteht aus einem konsistenten Netz aus zusammenhängenden dreidimensionalen Kästen, dieses Netz wird als *Grid* (Gitternetz) bezeichnet. In diesen *Grid*s befinden sich eigene virtuelle Welten. In Abbildung 4c ist ein Ausblick auf solch eine eigenständige Welt zu sehen. Nutzer können sich mit selbst erstellten virtuellen Avataren durch das *Grid* bewegen und mit Avataren anderer Nutzer, sowie mit der Welt in der sie sich befinden, interagieren und kommunizieren. In Abbildung 4b ist eine Gruppe von Nutzern zu sehen, welche über Avatare miteinander interagieren und kommunizieren. SL stellt Werkzeuge zur Verfügung um virtuelle Objekte zu erstellen. Diese virtuellen Objekte umfassen unter anderem Kleider und Accessoires für Avatare, sowie Orte, wie die Welt in Abbildung 4c und Gegenstände, wie zum Beispiel die Schiffe, welche in Abbildung 5b zu sehen sind. Da Rahmenwerke, welche für MRK-Szenarien verwendet werden, in der Lage sein sollten dreidimensionale Modelle ausreichend darzustellen, wird auch jeweils auf die Möglichkeiten der Erstellung, sowie die Unterstützung von dreidimensionalen Modellformaten eingegangen.

In SL können virtuelle Objekte aus *Primitives* bestehen oder als *Mesh* (Polygonnetz) importiert werden. *Primitives* können auch über UV-Maps in sogenannte Sculpted Objects (kurz Sculpties) umgewandelt werden.

Primitives (kurz Prim) sind virtuelle Objekte, die aus einfachen Formen, wie in Abbildung 5a, zusammengesetzt sind. Die einfachen Formen umfassen Würfel, Prisma, Kugel, Zylinder, Torus, Tube und Ring. Diese Prims können über Methoden wie Skalieren, Schneiden, Leeren und viele mehr verändert werden. In Abbildung 5b ist das rechte der beiden Schiffe durch Prims und Primoperationen erstellt worden. *Primitives* eignen sich besonders gut für Anfänger, da die Arbeit mit ihnen einfach und intuitiv ist. Außerdem eignen sie sich auch für



(a) *Second Life Primitives (Quelle: `http://secondlife.wikia.com/wiki/Prim`)*

(b) *Vergleich eines Schiffes erstellt aus Polygonen (links), mit einem Schiff erstellt aus SL Primitiven (rechts) (Quelle: `https://goo.gl/3xxeHV`)*

Abbildung 5: *Primitives (links) und ein aus Primitives erstelltes Schiff (rechts, das rechte Schiff), sowie ein Schiff erstellt aus Polygonnetzen (rechts, das linke Schiff)*

einfache Objekte oder zum Testen von Skripten, da für solche Aufgaben das Erstellen eines *Mesh* zu aufwendig wäre.

Sculpted Objects beschreiben ihre Geometrie über eine UV-Map. Eine UV-Map ist ein Bild bestehend aus Farbverläufen. Hierbei bestimmen die Intensität der Farben rot, blau und grün über den Abstand der Oberfläche des Objektes zum Ursprung. Sculpties eignen sich besonders für Objekte mit natürlichen Rundungen, werden jedoch kaum noch benutzt, da Polygonnetze meist einfacher zu erstellen und performanter bzgl. der Datenübertragung und des Renderings sind.

Polygonnetze sind ein Verbund von *Vertices* (Punkte), *Edges* (Kanten) und *Faces* (Flächen), welche zusammen ein *Mesh* (Polygonnetz) definieren. Das Erstellen von Polygonnetzen erfordert spezialisierte externe Programme, wie Blender oder Maya, welche meist viel Einarbeitungszeit erfordern, bieten hingegen aber auch eine größere künstlerische und gestalterische Freiheit beim Erstellen von virtuellen Objekten. In Polygonnetzobjekten können auch Texturen und Materialien, einzelnen Polygonen exakt zugeordnet werden. In Abbildung 5b auf der linken Seite ist solch ein virtuelles Polygonnetzobjekt zu sehen. SL unterstützt nur Polygonnetze im .dae Format.

Second Life verfügt über eine eigene Währung, die als *Linden Dollars* (L$) bezeichnet wird. Die *Linden Dollars* sind keine rein virtuelle Währung, da es möglich ist *Linden Dollars* mit echtem Geld zu kaufen. Es ist jedoch nicht möglich *Linden Dollars* in echtes Geld umzutauschen. Der Wechselkurs zwischen echtem Geld und L$ bezieht sich somit nur auf den Einkauf von L$ durch echtes Geld. Durch die Fähigkeit virtuelle Objekte für SL zu erstellen und das Vorhandensein einer eigenen virtuellen Währung, hat SL einen eigenen Wirtschaftskreislauf entwickelt. Bekanntestes Beispiel hierfür ist Ailin Graef mit ihrem SL Avatar *Anshe Chung*. Sie ist nach eigenen Angaben, der erste Avatar, der durch Geschäftstätigkeit innerhalb von SL im realen Leben Millionär wäre, wenn sie ihr Vermögen an L$ in SL in US$ umtauschen könnte [12].

[12] `https://goo.gl/ie2na8`

Second Life wird auch in Bildung und Forschung verwendet. Beispiele hierfür sind Sprachkurse [13], Kochkurse [14], das Durchführen von realen Experimenten in SL [15] sowie Forschung, Kollaborationen und Datenvisualisierung [6, 53] [16].

Linden Lab bietet Clients für Windows, macOS und die meisten Linux-Systeme. Die Clients rendern die dreidimensionalen Objekte von SL unter Benutzung von OpenGL Technologien. Außerdem gibt es mehrere vollwertige SL-Clients von Drittanbietern, welche gegenüber den offiziellen, von Linden Lab's angebotenen Clients, durch eigene Eigenschaften hervorstechen.

2.1.2 OpenSimulator

OpenSimulator erschien im März 2007 und ist eine quelloffene Serverplattform, mit welcher virtuelle Welten zur Verfügung gestellt werden können. Ein Entwicklungsziel des Systems war für lange Zeit die Unterstützung von *Second Life* (SL). Jedoch fiel im Januar 2015 die Entschieden, dass eine Kompatibilität mit SL kein Ziel der Entwicklung mehr darstellt, stattdessen wird es immer mehr in eine neue Basis zum Web3D umgebaut. *OpenSimulator* unterstützt sowohl Windows als auch Linux Betriebssysteme.

OpenSimulator verfügt über zwei Betriebsmodi. Der erste Betriebsmodus ist ein Alleinstehender, bei dem alle Aufgaben von der lokal vorhandenen Hardware übernommen werden. Dieser Modus ist gedacht zur Entwicklung von eigenen Anwendungen, kann jedoch auch genutzt werden, um einen einzelnen privaten Server zu betreiben. Der zweite Betriebsmodus von *OpenSimulator* ist der sogenannte *Grid-Modus*. Das *Grid* von *OpenSimulator* ähnelt dem *Grid* von SL, da es auch eine große zusammenhängende virtuelle Welt aufspannt, welche von mehreren Servern gemeinsam berechnet wird. Dieser Modus ist gedacht um größere virtuelle Welten darzustellen. Um die Arbeit der Server zu vereinfachen, unterstützt *OpenSimulator* mehrere Datenbankensysteme wie SQLite, MySQL, MariaDB und Microsoft SQL Server. Zusätzlich zu dem *Grit-Mode* existiert auch noch ein *Hypergrid* welches mehrere Server verbinden und es Nutzern ermöglicht, wie in SL, Welten von anderen Nutzern zu besuchen.

Da *OpenSimulator* viel von SL übernommen hatte, benutzte es auch lange Zeit die von Linden Labs entwickelte *Linden Scripting Language* (LSL). Während der Entwicklung von *OpenSimulator* wurden jedoch immer mehr von Nutzern gewünschte Funktionen zur LSL hinzugefügt. Somit wurde die LSL im Laufe der Entwicklung zur *OpenSimulator Scripting Language* (OSSL) weiterentwickelt.

OpenSimulator unterstützt ähnlich wie SL *Primitives*, *Sculpted Objects* und *Mesh* (Polygonnetz). Hierbei stellt auch *OpenSimulator* Werkzeuge zum Erstellen von *Primitives*, also Objekten bestehend aus grundlegenden und verformbaren Formen, wie Dreiecken, Würfeln und Zylindern, zur Verfügung. Das Arbeiten mit den *Primitives* ist intuitiv gestaltet, weshalb auch Anfänger vorzeigbare Objekte erstellen können. *Sculpted Objects* (kurz Sculpties) funktionieren in *OpenSimulator* ebenso wie in *Second Life*. Sculpties eignen sich daher besonders für Objekte mit natürlichen Rundungen, werden jedoch kaum noch benutzt, da Polygonnetze meist einfacher zu erstellen und performanter bzgl. der Datenübertragung und des Renderings sind. Diese Polygonnetze funktionieren genau wie in SL, haben jedoch auch die gleiche Beschränkung, nämlich die ausschließliche Unterstützung des .dae Formates.

[13] `https://goo.gl/1CSZyC`
[14] `http://www.vhs-sl.de/`
[15] `http://www.sltalk.de/index.php/2008/11/24/wenn-lernforschung-im-adventskalender-steckt/`
[16] `https://ccj.springeropen.com/articles/10.1186/1752-153X-3-14`

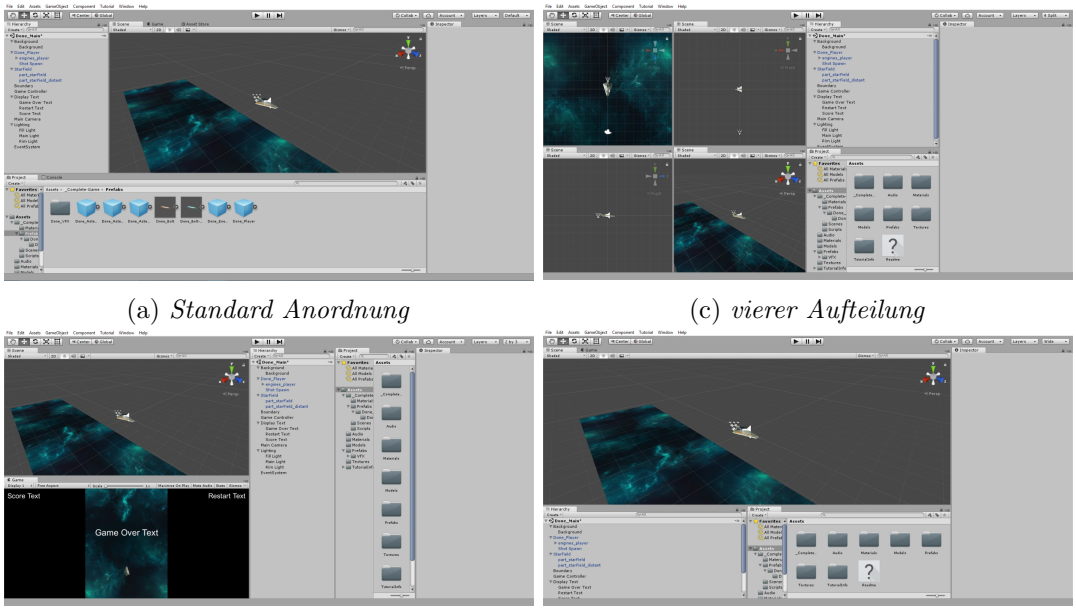
Unity3D ist eine Mehrzweck Laufzeit- und Entwicklungsumgebung für Spiele. Es wurde erstmals für OS X auf Apples *Worldwide Developers Conference* in 2005 angekündigt und am achten Juni des gleichen Jahres veröffentlicht. In 2009 auf der *Game Developer Conference* (GDC) wurde die Version 2.5 vorgestellt, die nativ auch unter Windows lauffähig ist. Der nächste große Schritt war die Version 3.0 welche am 27. September 2010 veröffentlicht wurde. Mit ihr wurden mehrere verschiedene Unity3D-Editoren für verschiedene Zielplattformen verworfen und in einen gemeinsamen Unity3D-Editor direkt integriert. Nutzer brauchten somit nicht mehr den Unity iPhone Editor um für iPhones zu entwickeln, sondern konnten den normalen Unity3D Editor benutzen um für Windows und iPhones zu entwickeln. Mit Unity3D Version 3 wurde es zur weitest verbreiteten Entwicklungsplattform für Unterrichtszwecke und die meistgenutzte Entwicklungsplattform für den Smartphone Markt [17]. Mit der Version 5.0 kamen, neben neuen Funktionen für den Editor, auch Änderungen zur Vertriebspolitik des Unity3D Editors. Seit 2009 gab es eine kostenlose Version welche Standardfunktionen zum Entwickeln von Spielen bietet und mehrere kostenpflichtige Versionen und kostenpflichtige Erweiterungen, welche dem Endanwender den vollen Funktionsumfang des Unity3D-Editors zur Verfügung stellten. Die unterschiedlichen Versionen umfassten (Stand Januar 2014):

- Unity (kostenlos)
- Unity Pro ($1.500)
- iOS Basic (kostenlos)
- iOS Professional ($1.500)
- Android Basic (kostenlos)
- Android Professional ($1.500)
- BlackBerry Basic (kostenlos)
- BlackBerry Professional ($1.500)
- Windows Store Basic (Enthalten in Unity Pro)
- Windows Store Professional (Enthalten in Unity Pro)

Ab der Version 5.0 wurde die kostenlose Version des Unity-Editors um einen Großteil der Funktionalität der vorherigen Pro-Version erweitert. Unterschiede zwischen der kostenlosen und der kostenpflichtigen Version beschränken sich seither größtenteils auf Lizenzrechte und Funktionen, welche nicht zum Erstellen von Produkten mit dem Editor benötigt werden [18]. Zum aktuellen Stand können mit Unity3D erstellte Programme auf bis zu 25 Plattformen portiert werden [19].

In Abbildung 6 ist der Unity3D-Editor in der Version 2017.1.0f3 zu sehen. Wie anhand der vier Bilder zu erkennen ist, ist der Editor in mehrere frei positionierbare und skalierbare Unterfenster aufgeteilt. Die am häufigsten benutzten Fenster sind *Project Browser*, *Inspector*,

[17] `http://www.marketwired.com/press-release/Unity-Technologies-Delivers-Unity-3-1325564.htm`
[18] `http://www.marketwired.com/press-release/unity-5-is-here-1997038.htm`
[19] `https://unity3d.com/de/unity/features/multiplatform`



(a) *Standard Anordnung*

(c) *vierer Aufteilung*

(b) *2 zu 3 Anordnung*

(d) *Weite Anordnung*

Abbildung 6: *Unity3D Editor in 4 vordefinierten Fensteranordnungen (Geladenes Projekt ist ein von UnityTechnology offiziell angebotenes und kostenloses Trainingsprojekt)*

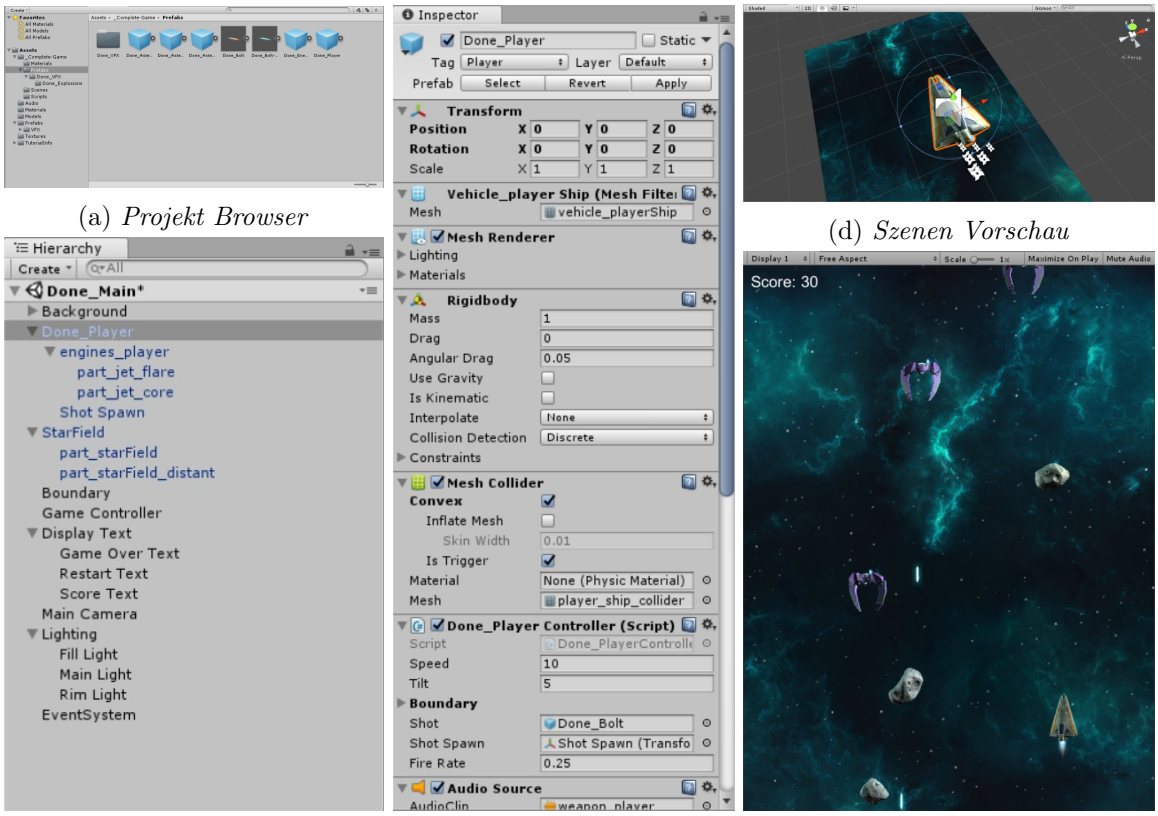
Game View, *Scene View* und *Hierarchy*, weshalb sie auch in jeder vordefinierten Anordnung des Unity3D-Editors 6 geöffnet sind.

Der Projekt Browser 7a listet alle Objekte des ausgewählten Projekts auf. Das Layout weist große Ähnlichkeiten zum *Finder* des Mac OS X und des Explorers von Windows auf. Durch die Ähnlichkeit zu bekannten Systemen soll es Nutzer ermöglicht werden sich schneller mit der Umgebung vertraut zu machen und die Arbeit mit diesen erleichtert werden.

In der Hierarchie 7b werden alle virtuellen Objekte angezeigt, welche sich in der ausgewählten Szene befinden. Die Liste wird automatisch aktualisiert, wenn ein Objekt in die Szene gebracht wird. Durch das Ziehen eines Objektes auf ein anderes, wird das erste dem zweiten untergeordnet. Wenn ein Objekt in der Hierarchie angewählt wird, werden im Inspektor weitere Details zu diesem Objekt angezeigt.

Zu angewählten virtuellen Objekten werden im Inspektor 7c weitere Informationen angezeigt, welche überprüft und verändert werden können. Hier werden alle Komponenten angezeigt, die einem Objekt angeheftet wurden. Komponenten die Objekten angeheftet werden können, umfassen unter anderem Skripte, Physik, Ton und Kollisionsobjekte. Zusätzlich werden hier auch öffentliche Variablen von angehefteten Skripten angezeigt und die Möglichkeit gegeben diese im Inspektor direkt zu verändern. Auf diese Art können Entwickler Werte verändern, ohne die entsprechenden Skripte direkt bearbeiten zu müssen. Somit wird eine interdisziplinäre Arbeit gefördert, da auch Nichtinformatiker Anpassungen vornehmen können, ohne sich mit Code auseinandersetzen zu müssen. Änderungen im Inspektor können auch direkt in der Szenen Vorschau beobachtet werden.

Die Szenen-Vorschau 7d wird verwendet um die virtuellen Objekte frei in einer dreidimensionalen Umgebung zu positionieren, um die Szene oder auch die Spielwelt zu gestalten.



(a) *Projekt Browser* (d) *Szenen Vorschau*

(b) *Hierarchie* (c) *Inspektor* (e) *Spiele Vorschau*

Abbildung 7: *Auswahl der meistgenutzten Unterfenster des Unity3D Editors*

Objekte können über *Drag and Drop* (Ziehen und Ablegen) von dem Projekt-Browser in die Szene gezogen werden und die Modelle in der Szene werden genau so über *Drag and Drop* bewegt. Durch Hilfsmittel wie *Grid Snapping* und 3D-Anfasser wird ein genaues positionieren der virtuellen Objekte ermöglicht. Um Objekte exakt zu positionieren, können auch die Positionswerte im Inspektor direkt gesetzt werden.

Über die Spiele-Vorschau 7e können die in der Szenen Vorschau erstellten Szenen direkt gestartet werden. Die Spiele Vorschau funktioniert nach dem *WYSIWYG* (*What you see is what you get* Sie bekommen was sie sehen) Paradigma. Somit ist es Entwicklern möglich ihr Produkt zu testen ohne das Projekt vorher zu bauen und die daraus entstehende ausführbare Programmdatei zu starten.

Der Unity3D Editor bietet auch Möglichkeiten, Projekte zu debuggen. Über das *Profiler* Fenster kann der Arbeitsaufwand des Systems zur Laufzeit des Projektes beobachtet werden. Hierbei kann zum Beispiel die CPU-Auslastung jedes einzelnen Skriptes, sowie der gesamte Speicherverbrauch beobachtet und kontrolliert werden.

Unity3D Unterstützt eine Vielzahl von externen Formaten für dreidimensionale Modelle [20], bietet jedoch, genau wie *Second Life* oder auch OpenSimulator, einige grundlegende Formen *Primitives*, über die simple Strukturen und Objekte erstellt werden können.

[20] `https://docs.unity3d.com/Manual/3D-formats.html`

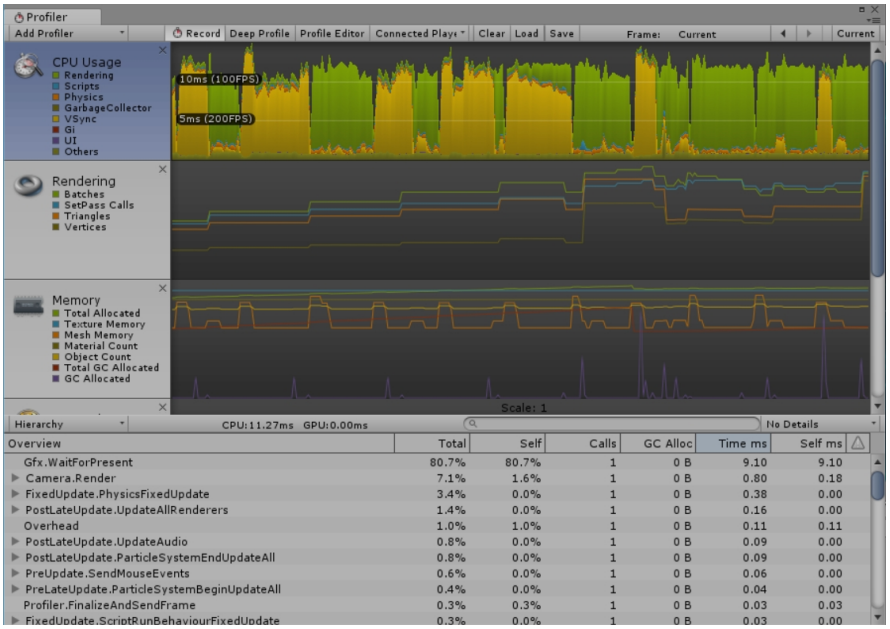


Abbildung 8: *Profiler*

2.1.3 C# Microsoft .Net

Ein Microsoft Team bestehend aus Anders Hejlsbergim, Scott Wiltamuth, und Peter Gold, begann 1999 die Arbeit an einer neuen Programmiersprache für die .Net Plattform, die von Microsoft entwickelt wurde. Die Sprache wurde ursprünglich als *Cool* (*C-like Object Oriented Language*) bezeichnet, wurde jedoch im Juli 2000 auf der *Professional Developers Conference* in Zusammenhang mit der .Net Initiative von Microsoft als C# vorgestellt. Im Entwicklungsprozess der Sprache C# waren die folgenden Ziele für das Design der Sprache festgelegt worden [18]:

- C# ist gedacht als eine simple, moderne, objektorientierte, allgemeine Programmiersprache.
- Die Sprache und ihre Implementierung soll für Softwareentwicklungsprinzipien, wie starke Typüberprüfung, Array Begrenzungsüberprüfung, Erkennung des Versuchs der Benutzung von uninitialisierten Variablen und automatische Speicherbereinigung Unterstützung bieten. Wichtig ist dabei die Robustheit, Stabilität und *programmer productivity* [21] der Software.
- Die Sprache ist vorgesehen um Softwarekomponenten zu entwickeln, welche für den Einsatz in verteilten Systemen gebraucht werden.
- Die Übertragbarkeit von Quellcode nach C# ist sehr wichtig, genau wie die Übertragbarkeit von Programmierer nach C# Projekte, insbesondere derer, die schon mit C und C++ vertraut sind.

[21] `https://en.wikipedia.org/wiki/Programming_productivity`

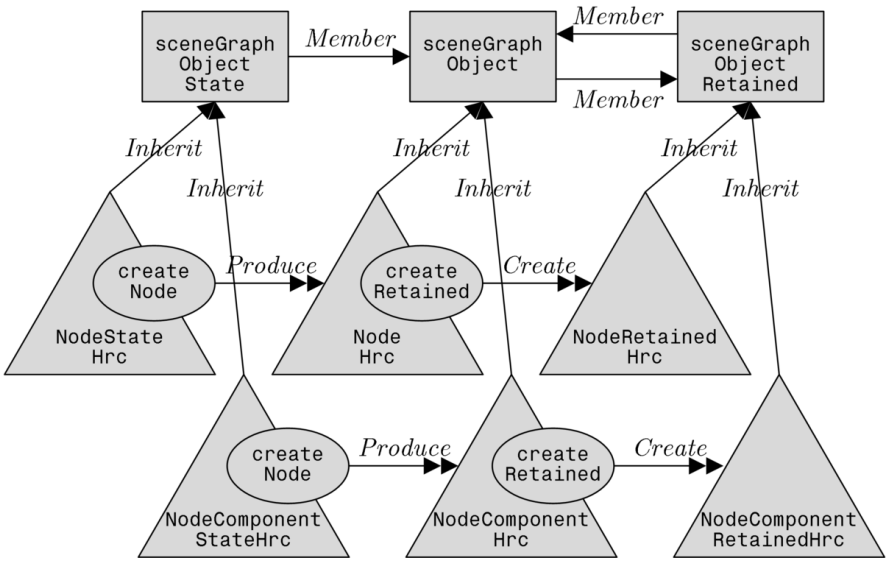


Abbildung 9: *Java 3D zentrale Hierarchie dargestellt in LePUS3 Quelle:* `https://goo.gl/uAJEKm`

- Unterstützung von Internationalisierungen ist sehr wichtig.
- C# ist angedacht nutzbar zu sein, um Anwendungen sowohl für gehostete als auch für eingebettete Systeme zu schreiben. Diese Systeme umfassen eine sehr große Bandbreite von sehr großen, ausgeklügelten Betriebssystemen, bis hin zu sehr kleinen dedizierten Funktionen.
- Obwohl C# Anwendungen ökonomisch in Bezug auf Speicher und Rechenleistungsanforderungen gedacht sind, ist es nicht das Ziel der Sprache, mit C oder anderer Assemblersprache in Bezug zu Laufzeit oder Größe zu konkurrieren.

C# wurde ursprünglich nur für Microsoft entwickelt, doch durch Xamarin und .Net Core ist es auch möglich C# für macOS, iOS und Android, sowie GNU/Linux und macOS zu entwickeln. Durch .Net Native und CoreRT ist es zusätzlich möglich, plattformunabhängigen Maschinencode für alle Plattformen mit C++ Schnittstelle anzubieten.

2.1.4 Java 3D

Intel, Silicon Graphics, Apple, und Sun planten und entwickelten im Jahr 1996 unabhängig voneinander eine *retained mode* Szenengraphenanwendung. Da sie dies alle mit Java umsetzen wollten, entschieden sich die Unternehmen für eine Zusammenarbeit. Das Ergebnis der Entwicklung ist Java 3D. Die erste veröffentlichte Version von Java 3D war eine öffentliche Beta im März 1998 und die finale Veröffentlichung fand im Dezember 1998 statt. Java 3D lief unter OpenGL und Direct3D. Seit Version 1.6.0 läuft Java 3D nur noch unter JOGL.

Die Weiterentwicklung von Java 3D wurde Mitte 2003 eingestellt und im Sommer 2004 als offenes Gemeinschaftsprojekt weitergeführt. Am 29.01.2008 wurde die Weiterentwicklung von Java 3D zugunsten von der Entwicklung eines 3D Szenengraphens für JavaFX verschoben. Seit dem 28.02.2008 wurde der gesamte Quellcode von Java 3D unter GPL version 2 Lizenz mit GPL Ausnahmegenehmigung fürs Linken veröffentlicht.

2.2 Verwandte Systeme

In diesem Abschnitt werden Arbeiten vorgestellt, welche die Systeme aus dem vorherigen Abschnitt als Rahmenwerke benutzten. Am Ende von diesem Kapitel werden die Arbeiten zusammengefasst und verglichen.

2.2.1 Darstellung von Kontextfeldern (Lifton 2007)

Lifton stellt als eine Motivation für seine Arbeit das alte John Hancock Berkley Gebäude aus Boston vor [22]. Markenzeichen dieses Gebäudes ist ein Leuchtfeuer, welches regelmäßig die Blicke der in Boston lebenden Menschen auf sich zieht, da es Wettervorhersagen signalisiert. Dieses Leuchtfeuer, so argumentiert Lifton weiter, ist ein gutes Beispiel dafür, wie die visuelle Repräsentation von einfachen Sensordaten, eine Räumlichkeit lebendiger gestalten kann. Des Weiteren argumentiert er, dass entsprechende Visualisierungen von Datenströmen in der virtuellen Welt interessanter sind, da sie zum einen kostengünstiger und einfacher umzusetzen sind als in der echten Welt und zum anderen, in großen virtuellen Welten, ein größeres Bedürfnis existiert, diese Welten lebendiger zu gestalten.

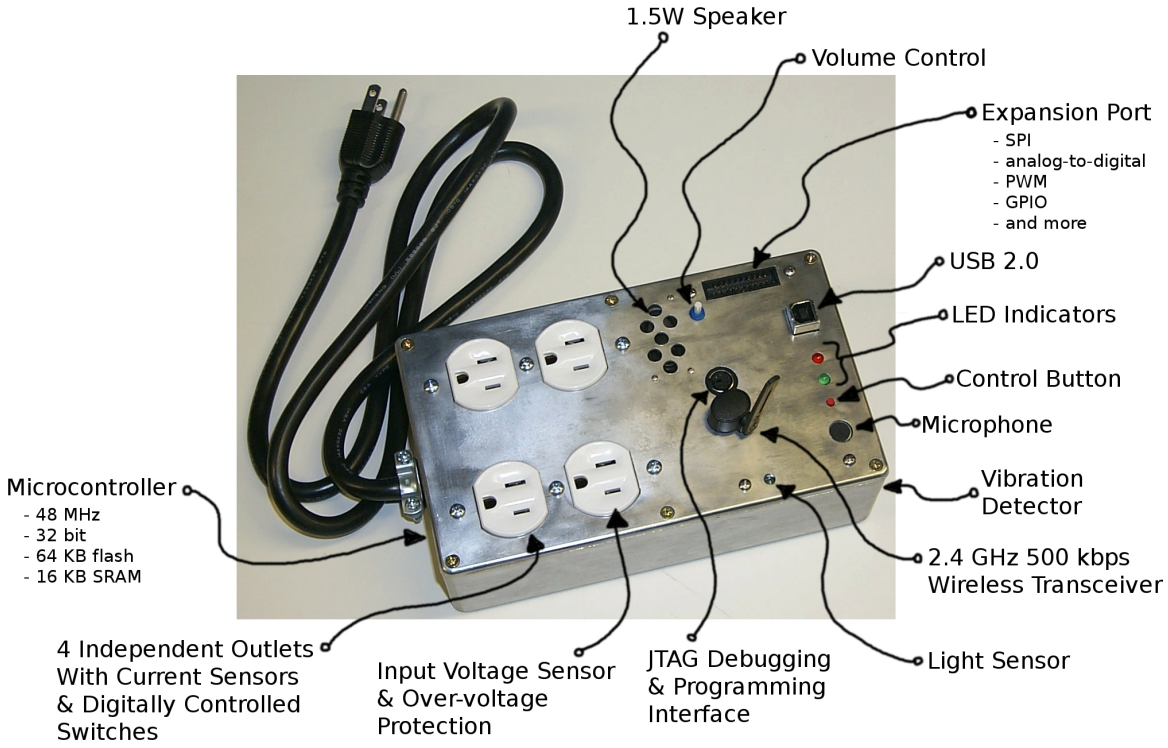


Abbildung 10: Plug Steckerleiste mit Anschlüssen (Quelle: [34])

Für seinen Ansatz konnte Lifton nicht die bestehenden Sensoren, eines Smartphones oder Laptops benutzen, da diese entwickelt wurden, um unabhängig von anderen Systemen arbeiten zu können. Lifton hingegen brauchte für seinen Ansatz ein Netzwerk von Sensoren, welches zusammenarbeitet. Somit entwickelte er sein eigenes Sensornetzwerk bestehend aus

[22] https://en.wikipedia.org/wiki/Berkeley_Building

einer physischen, instrumentieren Steckerleiste (Plug) in einer sicheren Aluminiumbox, welche mit diversen Sensoren und Handlungsobjekten ausgestattet sein kann. Diese Steckerleisten enthalten standardmäßig vier Stromsteckdosen, welche durch Sensoren den Stromfluss überwachen, sowie Schalter, welche die Stromzufuhr durch die Steckdosen virtuell kontrolliert kappen können. Weiterhin wurden Sensoren verbaut, welche den Lichteinfall messen, Erschütterungen erfassen, Geräusche wahrnehmen und die Temperatur messen können. Als weitere Eingabemöglichkeit wurde auch noch ein frei programmierbarer Knopf auf dem Gerät verbaut. Zusätzlich sind noch Lautsprecher und LED-Leuchten an dem Gerät angebracht, welche zur Ausgabe von Informationen in die echte Welt benutzt werden können. Zur Programmierung der Instrumente und zur Fehlersuche gibt es eine Schnittstelle sowie USB 2.0 Anschlüsse. Die Firmware, die das Plug verwendet, wurde von Lifton selbst geschrieben und auf die Bedürfnisse seiner Arbeit hin optimiert. Der Aufbau solch eines Gerätes wird in Abbildung 10 dargestellt.

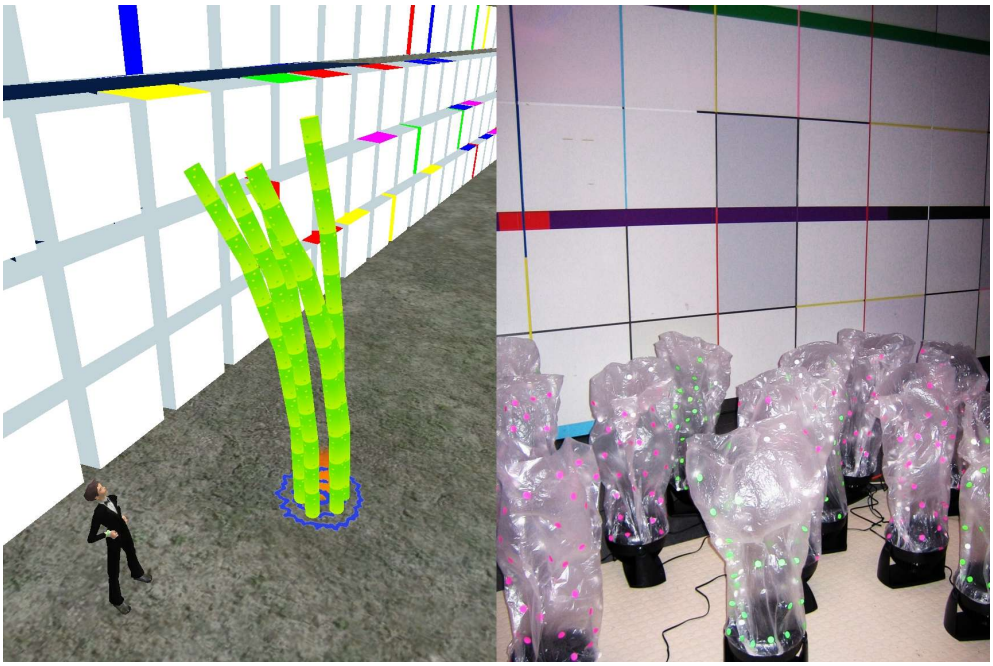


Abbildung 11: *Virtuelle Repräsentation des Datenteichs (links) sowie mehrere reale Gegenstücke zum Datenteich (rechts) [33]*

Lifton hat sich für *Second Life* [23] als virtuelle Welt entschieden, in der er seine Plugs und deren Sensormessungen visualisiert. Hierfür schuf er sogenannte Data Ponds (Datenteiche). Ein Datenteich besteht aus einem kleinen Teich aus dem Stängel hervorragen, sowie ein schwebendes Feuer direkt über dem Teich 11. Ein Datenteich visualisiert in der virtuellen Welt die Messungen einer Plug Steckerleiste auf folgende Art und Weise:

- Die Anzahl der Ringe auf der Haut der Stängel entspricht dem größten Lichtlevel der aktuellsten Sekundenmessung des Plugs.

[23] `http://secondlife.com`

- Die Farbe der Stängel variiert linear von blau über gelb zu Grün mit der gemessenen Temperatur in einem Radius von 18 bis 29 Grad Celsius.
- Die Stängel schwingen gemütlich, wenn das Plug keine Bewegung wahrnimmt und hektisch, wenn Bewegungen gemmessen werden.
- Die Größe der Pfütze ist proportional zum gemessenen Lärmpegel.
- Die Größe und Intensität der Flamme ist proportional zum gemessenen totalen durchschnittlichen Stromfluss durch das Plug Gerät.

Laut Liftons Definition einer Dual Reality, besteht diese nicht nur aus Einflüssen der echten Welt auf die virtuelle Welt, sondern auch aus virtuellen Gegebenheiten, die Einfluss auf die echte Welt ausüben. So hat er normale Ventilatoren mit einer durchsichtigen Plastikfolie überspannt und sie mit den Plug Steckerleisten verbunden. Die Intensität des Ventilatorgebläses kann nun durch die Plug Steckerleiste gesteuert werden. So kann die Blasintensität eines Ventilators, zum Beispiel den Abstand eines Avatars zu dem entsprechenden Datenteich widerspiegeln, womit auch die virtuelle Welt Einfluss auf die echte nehmen kann.

Wie schon erwähnt, setzt Lifton für sein Projekt sowohl auf eine eigens programmierte Firmware für die Plug Steckerleisten, welche in der echten Welt positioniert sind und per Sensoren Messungen vornehmen, als auch auf *Second Life* für den digitalen Teil der Arbeit. *Second Life* verfügt über eine eigene Skriptsprache mit Namen LSL (Linden Scripting Language) welche der vorgesehene Weg ist mit *Second Life* zu kommunizieren und dementsprechend auch von Lifton zur Kommunikation zwischen Plug und seiner virtuellen Repräsentation verwendet. Da die Programmierung des Plugs für diese Arbeit nicht von Bedeutung ist, wird darauf nicht weiter eingegangen.

2.2.2 Twin World Mediator (Brandherm 2008)

Ausgangspunkt für Brandherm ist ein Positionierungssystem für Innenräume, welches er und seine Kollegen installiert haben [7]. Das System nutzt fest installierte Sender und mobile Empfänger. Die Sender sind sowohl RFID-Tags als auch Infrarotbaken, welche Sensordaten ausstrahlen. Ein Nutzer solch eines Systems ist mit einem eigenen mobilen Gerät ausgestattet, welches anhand der empfangenen Sensordaten, bestehend aus Position, Ausrichtung und Signalstärke, die eigene Position, durch einen entsprechenden Positionierungsalgorithmus berechnet. Nach dieser Berechnung wird die eigene Position auf einer zweidimensionalen Karte des mobilen Gerätes angezeigt. Hierbei hängt die Genauigkeit der Positionierung von mehreren Faktoren, wie verfügbaren Sensoren, Anzahl der Sensoren, Platzierung der Sensoren, Infrastruktur und Interferenzen der einzelnen Sensoren, ab, was den Aufbau des Systems zu einem nicht trivialen Problem macht. Da die Installation jedoch ungenau war und Versuche die Genauigkeit des Systems durch mehr Sender zu erhöhen scheiterten, entschied sich Brandherm und sein Team ein bidirektionales Interface für *Second Life* (SL), den **Twin World Mediator**, zu programmieren [8].

Brandherm erfüllt mit mit dem *Twin World Mediator* drei Ziele. Das erste Ziel ist es, ein generisches System zu entwickeln, welches genutzt werden kann, um eine verbesserte Genauigkeit des vorher beschriebenen Positionierungssystems zu erreichen. Zusätzlich soll es genutzt werden können, um andere Sensornetzwerke und Systeme, auf einfachstem Wege auf verschiedene Einstellungen und Anordnungen im dreidimensionalen Raum, zu testen. Das zweite Ziel des *Twin World Mediator* ist es einen direkten bidirektionalen Rückmeldekreislauf zu bieten.

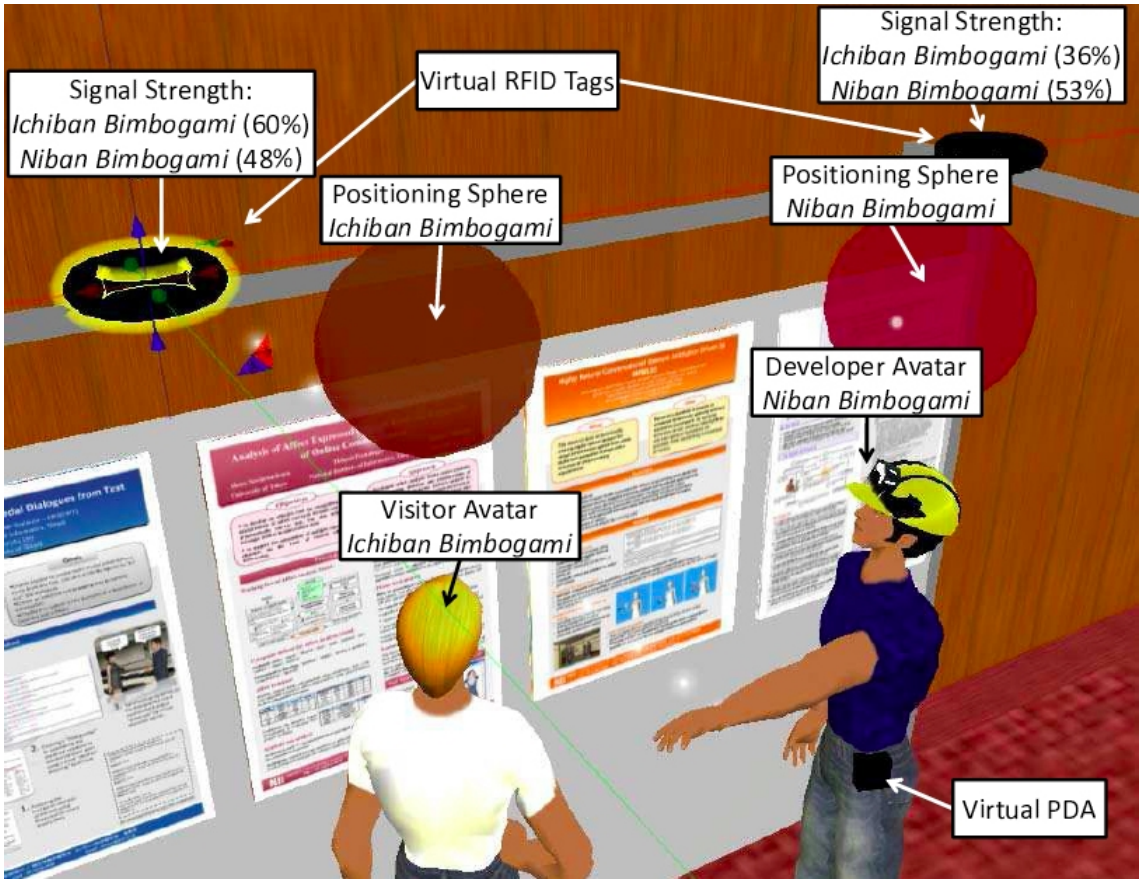


Abbildung 12: *Beispiel des Twin World Mediator zum Positionierungssystem in Second Life mit einem Besucher Avatar und einem Entwickler Avatar [43]*

So führt die Änderung eines echten Gerätes, direkt dazu, dass sich sein virtuelles Gegenstück in *Second Life* an die Änderung des realen Gerätes anpasst, entsprechend führt die Veränderung eines virtuellen Gerätes dazu, dass sich sein reales Gegenstück auch verändert. Das dritte Ziel ist ein Bezug zur dritten Dimension, da laut Brandherm die Position und Orientierung eines Sensors entscheidend für das Arbeiten eines Systems sein kann. In Abbildung 12 ist die fertige Implementierung von Brandherms *Twin World Mediator* zu sehen, welche genutzt wird, um das am Anfang angesprochene Positionierungssystem zu testen. Hierbei wird in *Second Life* der Aufbau des Positionierungssystems samt Sender und Empfänger nachgestellt. Um die mobilen Empfänger zu simulieren, müssen Nutzer in *Second Life*, ihre Avatare mit einem virtuellen PDA ausstatten. Da *Second Life* nicht über eine ausreichende Simulation von Infrarotstrahlen, oder RFID Tags verfügt, wird ein Simulationsalgorithmus benutzt. Dieser Simulationsalgorithmus erhält als Input die Positions- und Orientierungsdaten der virtuellen PDAs der Avatare, sowie die Positions- und Orientierungsdaten der Sensoren. Mit dem Input werden die Sensordaten berechnet, die bei den PDAs ankommen. Anhand dieser berechneten Daten wird dann die Position der Avatare in *Second Life* berechnet und in Form von transparenten Kugeln angezeigt. Da Brandherm, wie auch schon Lifton, *Second Life* für sein System benutzt, wird auch hier die *Lindon Scripting Language* zur Kommunikation mit *Second Life* genutzt. Der *Twin World Mediator* übernimmt jedoch nicht nur die Aufgabe des Übermitt-

lers, sondern auch die einer Datenbank. So werden alle statischen Objektinformationen bei der Initialisierung einer Szene mitgeteilt und von der Datenbank abgespeichert um Bandbreite und Ressourcen zu schonen. Nicht statische Objekte hingegen senden in regelmäßigen Abständen ihre Position, Orientierung und weitere variable Informationen, wie zum Beispiel bei den virtuellen RFID-Tags, ihre momentane Signalstärke, über einfache SLS Skripte [43].

2.2.3 Eolus One (Coleman 2009)

In der Schweiz wurde mit *EolusOne* eine X-Reality Plattform, zur Überwachung und Kontrolle von Gebäude-basierten Sensordaten nach *Second Life*, geschaffen. Neben *Second Life* wurde ebenfalls die kostenlose Alternative *OpenSimulator* unterstützt, welche als eigene Anwendung auf einem Server im Intranet laufen kann. Durch den Betrieb im Intranet wird der Zugriff unbefugter erschwert und die Sicherheit folglich erhöht. *EolusOne* erforscht mit seinem Ansatz zwei Philosophien der x-Reality. Zum einen hat es eine zentrale, jedoch nicht geografisch gelegene Kontrollstation, zum anderen werden Informationen als interaktive dreidimensionale Simulation dargestellt. Außerdem wurde es genutzt um die Hypothese zu testen, ob ein virtuelles Netzwerk in der Lage ist für größere Effizienz im Immobilienwesen, in der Systeminstandhaltung und während der Arbeitszeit, zu sorgen. Hierzu wurden zum einen in Gebäuden verschiedenste Sensoren montiert, welche von der Raumtemperatur, über Öffnungsstatus von Türen, bis hin zum Lichteinfall, verschiedenste Faktoren messen und überwachen können. Zudem wurden Schnittstellen geschaffen um Gerätschaften innerhalb des Gebäudes, wie Heizung, oder Klimaanlagen, zu überwachen und zu bedienen. Die erfassten Daten werden als Events empfangen und mittels der dreidimensionalen Darstellung von *Second Life*, oder auch *OpenSimulator* in einer Art virtueller Kommandozentrale gebündelt wiedergegeben, wie in Abbildung 13 zu sehen ist.

Angefangen hat das Projekt mit der Modifikation eines Puppenhauses, in welchem Lichter, über ein virtuelles Kontrollzentrum, an und wieder ausgeschaltet werden konnten. Nach weiteren Untersuchungen zu Gebäudesystemen, wurde auch ein Gebäudeteil der Entwicklungsfirma mit entsprechenden Systemen ausgestattet, welche auch Zugriff auf Fahrstuhlsteuerung, Lichter, Klimaanlagen, Brandschutzsysteme und vieles mehr boten. In dem virtuellen Kontrollzentrum kann per virtuellen Avatar auf den Zustand der einzelnen Systeme zugegriffen und mit den entsprechenden Zugriffsrechten auch Änderungen an den Systemen vorgenommen werden oder, wenn erforderlich, Personal zur Behebung von Problemen geschickt werden.

2.2.4 Virtuelle Schokoladenfabrik (Back et. al 2010)

Ein ähnliches Konzept zu einer Gebäudeüberwachung von *Eolus One*, verfolgt auch die Virtuelle Schokoladenfabrik [2]. In der Planungsphase der Schokoladenfabrik wurde beschlossen eine moderne Fabrik zu bauen, in der Schokolade auf einem hohen technologischen Niveau hergestellt werden kann. Somit wurden Sensoren und Kameras schon bei dem Bau der Fabrik eingeplant und installiert, um die Vorgänge, in der Fabrik und dem dazugehörigen Labor, verfolgen und überwachen zu können. Die fertiggestellte Fabrik wiederum wurde virtuell nachgebaut und in *Multiverse*, einer 3D-Plattform für Onlinespiele, welche im Jahr 2011 eingestellt wurde, dargestellt, wie in Abbildung 14 zu sehen ist. Ähnlich wie in Eolus One ist es möglich sich mit virtuellen Avataren durch die virtuelle Darstellung der Fabrik begeben und die Sensordaten und Informationen, aus der echten Fabrik in der virtuellen Darstellung abru-

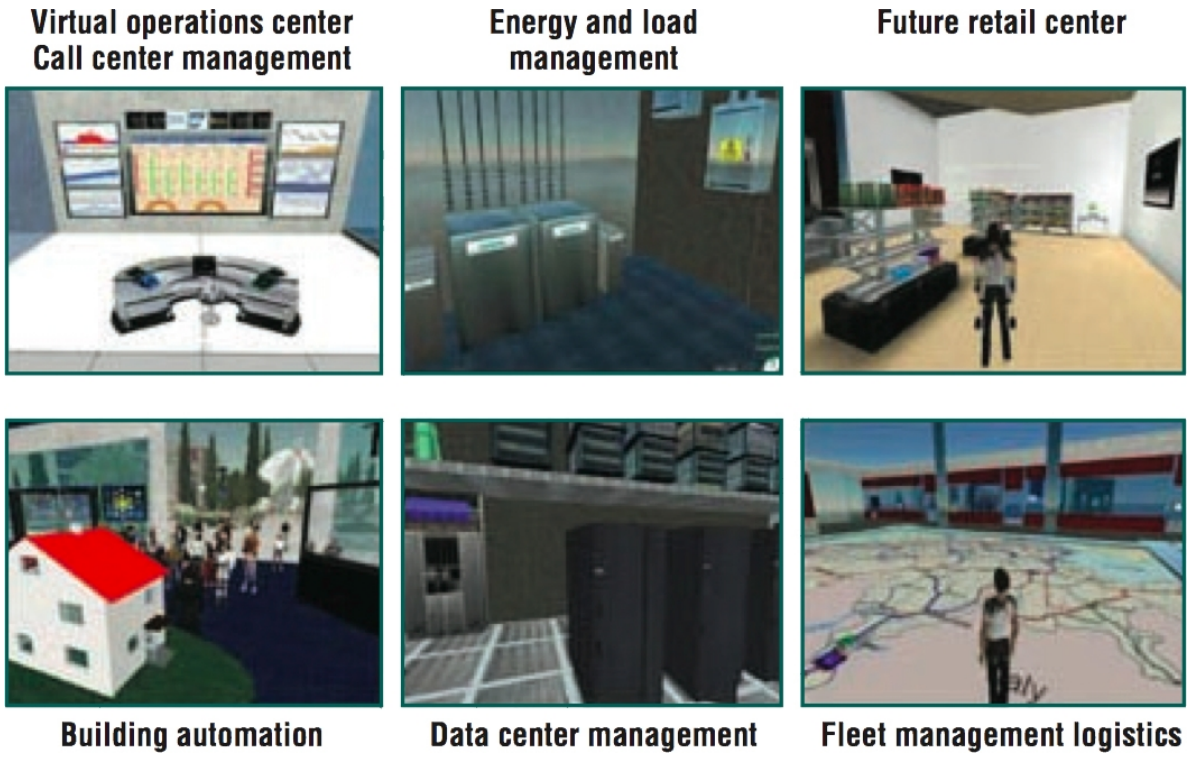


Abbildung 13: *EolusOne[12]*

fen und wenn nötig, Änderungen an dem zurzeit laufenden Maschinenaufgaben vornehmen. Zusätzlich können sich Nutzer in der virtuellen Welt, eine live Wiedergabe aus der echten Fabrik ansehen, indem sie sich mit ihrem Avatar dicht an eine Maschine begeben. Somit ist es sowohl Meistern möglich, ihre Lehrlinge zu unterstützen, als auch zum Beispiel Managern und Operatoren, tägliche Aufträge zu besprechen.

Die virtuelle Fabrik ist auch der Öffentlichkeit für Besichtigungen und virtuelle Rundgänge zugänglich. Damit jedoch Kunden, oder Besucher keinen Schaden, durch falsche Einstellungen an den Maschinen, anrichten können, wurden unterschiedliche Zugriffsrechte für Avatare erteilt. Um die virtuelle Fabrik zu erweitern, wurde auch noch eine Begleitapp für Smartphones entwickelt. In der Begleitapp ist es, wie in der virtuellen Fabrik, möglich Sensorinformationen der Maschinen abzurufen und Live Übertragungen der Maschinen zu betrachten, oder sogar einzelne Maschinen zu steuern.

Die virtuelle Fabrik wurde 2009 in Betrieb genommen und seither in verschiedenen Iterationen weiterentwickelt. Die Entwickler setzten anfangs noch auf die Multiverse Engine [24], eine für *Massive Multiplayer Online Games* (MMOGs) ausgelegtes Start-up, welches 2011 aufgrund von ausbleibenden Profiten, geschlossen wurde. Die virtuelle Schokoladenfabrik läuft mittlerweile auf der Unity3D-Engine [1].

[24] `https://en.wikipedia.org/wiki/Multiverse_Network`

Abbildung 14: *Die Schokoladenfabrig links real und rechts virtueller Nachbau [3]*

2.2.5 Yamamoto (Stahl und Haupert 2006, Stahl 2009)

Der *BMW Personal Navigator* (BPN) ist eine Navigationshilfe, die nicht nur im Straßenverkehr bei der Wegfindung hilft, sondern auch abseits der Straßen, auf Wegen und in Gebäuden [30]. Ein Nachteil dieser Anwendung ist jedoch, dass die Wegfindung auf einem Graphenmodell basiert, was dazu führen kann, dass die Route für einen Fußgänger umständliche Umwege annimmt, da im Modell kein direkter Pfad vorhanden ist. Zusätzlich ist die Darstellung und entsprechend die Wegfindung in Gebäuden im System nicht optimal gelöst. Dies diente als Motivation für *YAMAMOTO* (Yet Another Map Modelling Toolkit), welches genau wie BPN eine universale Navigationshilfe für den alltäglichen Gebrauch ist, jedoch auch Gebäude zuverlässig und einfach darstellen kann [51, 49].

Abbildung 15: *YAMAMOTO Bedienoberfläche mit Erklärungen [49]*

Hierzu ist es möglich in Gebäuden Räume, Flure und verschiedene Arten von Stockwerkübergänge wie Treppen oder Fahrstühle zu modellieren und entsprechend im Modell zu annotieren. Außerdem können Gebäude auch in unterschiedliche Ebenen, samt Übergängen, aufgeteilt werden, was die Wegfindung in und das Modellieren von mehrstöckigen Gebäuden vereinfacht. Um nicht alles von Hand modellieren zu müssen, ist es möglich, Grundrisspläne für Gebäude und geografische Karten für Umgebungen als Bild einzulesen und entsprechend um den benötigten Maßstab des Modells zu rotieren und zu skalieren. In YAMAMOTO werden die virtuellen Karten mit Polygonen dargestellt, welche ihre Kanten mit den Nachbarpolygonen teilen. Jedes Polygon, bestehend aus *Vertices* (Punkten), *Edges* (Kanten) und *Faces* (Flächen), enthält spezifische Informationen über die Fläche in der echten Welt, die es repräsentiert. Die Information für solch eine Fläche ist zum Beispiel, ob eine Person die Fläche durchqueren kann oder nicht. Die Kanten der Polygone enthalten noch zusätzliche Informationen, die ihre Durchlässigkeit bestimmen. So kann eine Kante eines Polygons eine undurchlässige Mauer, ein Fenster oder eine Tür sein und zusätzlich enthält sie noch Informationen, in welche Richtung sie durchquert werden kann. Über Polygone können auch verschiedene Ebenen eines Gebäudes verbunden werden, hierzu werden Polygone als Treppe, Schräge, oder Aufzug modelliert. Aufgrund der entstandenen Polygonkarte und den Verbindungseigenschaften der einzelnen Polygone erstellt YAMAMOTO einen Verbindungsgraphen. Dieser Verbindungsgraph kann genutzt werden, um mit einem A*-Algorithmus einen beliebigen begehbaren Pfad von einem Punkt A zu einem Punkt B zu berechnen, wie in Abbildung 16 zu sehen ist.

YAMAMOTO bietet jedoch mehr Funktionalität als die, eines reinen Routenplaners. So können Möbelstücke und andere Objekte durch ineinander geschachtelte, texturierte und verformte virtuelle Boxen dargestellt werden. Diese Modelle werden in einem yml-Format, welches sich an XML orientiert, abgespeichert. Ein Vorteil von yml, gegenüber Standard XML, ist, dass es auch von Menschen gelesen und verändert werden kann. Diese Modelle können zusätzlich in die beiden Formate VRML (Virtual Reality Modeling Language) [25] und RoSiML (Robot Simulation Modeling Language) [26] konvertiert und exportiert werden. Somit können die Inhalte auch beispielsweise auch auf mobilen Endgeräten genutzt werden. Weiterführend können Schnittstellen von YAMAMOTO genutzt werden, um Sensoren und Aktuatoren aus der echten Welt mit einzubinden, wie in Abbildung 17 zu sehen ist.

Um Interaktionen, wie in Abbildung 17 beschrieben, zu ermöglichen, benutzt YAMAMOTO einen *Universal Control Hub* (UCH) [20, 56] Dieser UCH ist über die ISO [25] definiert und beschreibt die Erreichbarkeit von Controllern, inklusive geeigneter Benutzeroberflächen und Benutzerschnittstellen, für verschiedene Systeme. Schlussendlich ist auch angedacht das Konzept der *Synchronisierten Realitäten* [50] mit YAMAMOTO umzusetzen, welches eine Synchronisation zwischen mehreren echten Räumlichkeiten vorsieht. So soll das System eine soziale Verbindung von Benutzern darstellen, indem physische Umgebungen in Bezug auf Licht, oder Fernsehprogramm synchronisiert werden.

YAMAMOTO selbst wurde mit C# auf Microsofts .NET Plattform implementiert. Zur Kommunikation zwischen einem mobilen Nutzer und dem System können RFID Tags, Bluetooth und zum Beispiel Infrarotbaken genutzt werden, welche von Nutzern im System positioniert werden können.

[25] `https://de.wikipedia.org/wiki/Virtual_Reality_Modeling_Language`
[26] `http://www.informatik.uni-bremen.de/spprobocup/RoSiML.html`

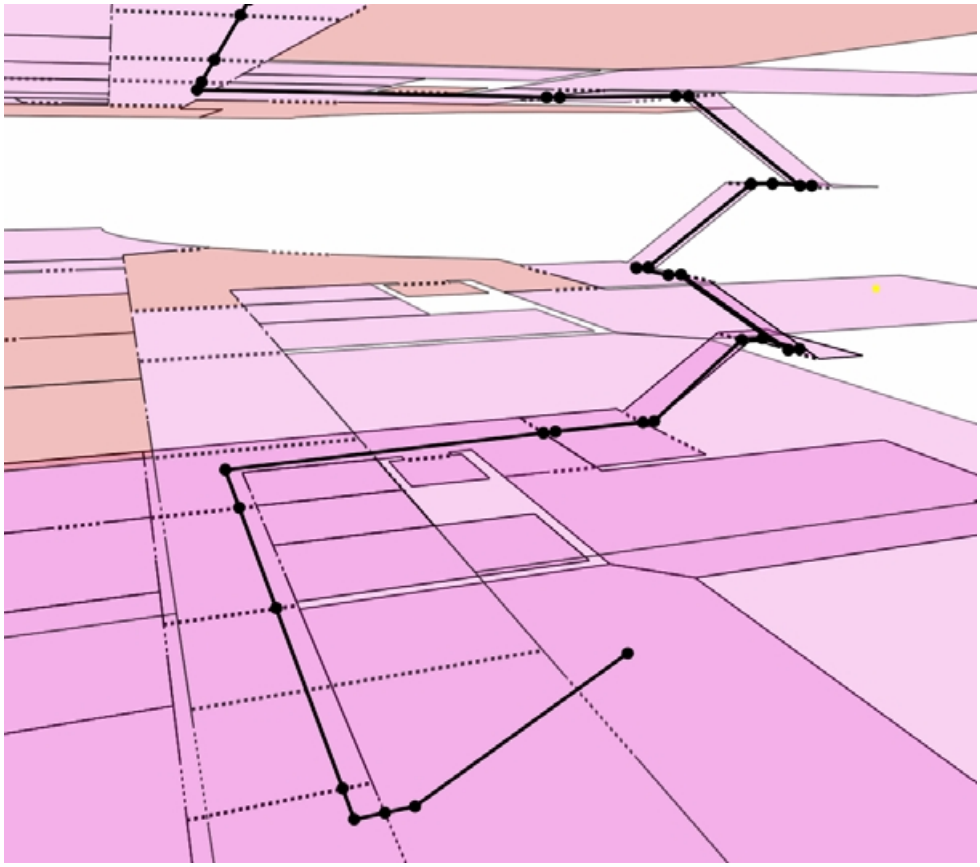


Abbildung 16: *Beispiel eines geplanten Pfades durch ein Gebäude unter Benutzung von Treppen [51]*

2.2.6 DuRMaD (Kahl 2014)

Kahl hat ein *Dual Reality* Rahmenwerk entwickelt, welches durch die Verbindung einer interaktiven 3D-Visualisierung einer Cyber-Physical Environment (CPE zu deutsch Cyber-Physische Umgebung) mit einer modularen Kommunikationsinfrastruktur das Monitoring und die Steuerung dieser CPE ermöglicht [27]. Dieses Rahmenwerk besteht aus dem eigens entwickelten *Dual Reality Management Dashboard* (DuRMaD), sowie einer dafür speziell entwickelten eventbasierten Kommunikationsinfrastruktur, welche als *Event Broadcasting Service* (EBS) bezeichnet wird.

Eine Smarte Umgebung bietet die Möglichkeit Informationen über sich selbst zu erlangen und anzuwenden, um sich ihren Bewohnern anzupassen und so die Erfahrung der Nutzer inerhalb dieser Umgebung zu verbessern [13]. Ein *Cyber-Physisches System* (CPS) ist eine erweiterte Form einer Smarten Umgebung durch das Verknüpfen der Sensor-Aktuator Netzwerke einer Smarten Umgebung mit dem Cyberspace. Somit können die erfassten Daten dieser Umgebung über weltweit verfügbare Informationen ausgewertet werden. Wenn sich nun alle in einer Umgebung befindlichen CPS zusammenschließen, ergibt dies eine CPE. DuRMaD wurde entwickelt, um eine interaktive 3D-Darstellung solch einer CPE zu ermöglichen. Hierzu wird von DuRMaD automatisch, basierend auf einem Grundrissplan und vorher modellierten dreidimensionalen virtuellen Modellen, ein Umgebungsmodell erstellt. Zusätzlich ermöglicht

Abbildung 17: *Das Fernsehprogramm in der echten Welt wird, genau wie das Licht über dem Herd, von einem virtuellen Gegenstück in der virtuellen Welt (Laptop im Vordergrund) gespiegelt. Außerdem kann der Fernsehkanal über die virtuelle Darstellung geändert werden: [50]*

es die Bereitstellung aller nötigen Funktionalitäten, um sowohl mit der CPE interagieren zu können, als auch die CPE selber zu managen. Damit DuRMaD all diese Funktionalitäten liefern kann, besteht es aus fünf, miteinander kombinierten, Bereichen. Diese sind:

- Datenquellen mit Informationen über die CPE
- spezifische Schnittstellen zum Empfangen und Verwenden von externen Informationen und Daten
- Kernmodule enthalten grafische Repräsentation und Interaktionsfunktionalität
- Kontroll- und Steuerungssysteme um eine Verbindung zwischen realer und virtueller Repräsentation zu bilden
- Schnittstellen für externe Verarbeitungsautomatismen um Zugriff auf DuRMaD interne Daten zur weiteren Verarbeitung

Zusätzlich wurde DuRMaD in Java3D implementiert, um einen Plattform unabhängigen Einsatz zu ermöglichen.

Um eine bestmögliche Kommunikation zwischen CPE und DuRMaD zu ermöglichen, wurde die Kommunikationsinfrastruktur *Event Broadcasting Service* (EBS) entwickelt. Wichtig für die Kommunikation zwischen einer CPE und einem entsprechenden virtuellen Rahmenwerk, zur Kontrolle und zum Managen, entsprechender CPE ist, dass eine Synchronisation, sowie ein bidirektionaler Datenaustausch gewährleistet werden kann. Besondere Berücksichtigung

Abbildung 18: *DuRMaD in Java3D-Version [27]*

benötigt hierbei das Konzept der Erweiterten Dual Reality (DR++). Das Konzept der Erweiterten Dual Reality besagt, dass eine Simulation auf einer CPE, verschmolzen mit ihrer Virtualisierung, Einfluss auf die CPE, die Virtualisierung, oder beide nehmen kann, was zu einem Verlust der Synchronisation zwischen CPE und Virtualisierung führen kann. Gleichzeitig hält die Simulation der CPE jedoch die Möglichkeit bereit, die reale Umgebung auf den Zustand vor der Simulation zu bringen und eine erneute Synchronisation zwischen CPE und Virtualisierung zu gewährleisten. Zusätzlich bietet das EBS noch Filter und Vorverarbeitungsmodule, welche vor Versand von Daten und Informationen, sowie nach deren Empfang, verwendet werden können.

Durch den Zusammenschluss von DuRMaD und EBS wurde somit eine Kommunikationsinfrastruktur geschaffen, die es ermöglicht, CPEs zu überwachen, zu steuern und gleichzeitig Simulationen in diesen zu integrieren [27]. Integriert und getestet wurde DuRMaD in einer Laborumgebung, die einem Supermarkt nachempfunden wurde. In 18 die von DuRMaD dargestellte Supermarktumgebung zu sehen.

2.3 Zusammenfassung

In diesem Kapitel wurden mehrere Rahmenwerke, sowie Systeme welche diese Rahmenwerke, unter Benutzung von Dual Reality Paradigmen verwenden, beschrieben. Entsprechend der individuellen Ziele, die jedes einzelne dieser Systeme verfolgt, besitzt es auch eigene Vorzüge, im jeweiligen Anwendungsbereich. Der in dieser Arbeit entwickelte TEDURU-Server wurde ebenso gemäß spezialisierten Anforderungen entwickelt. Als Anforderung für den TEDURU-Server wurde die Verwendung vom *network-toolkit TECS* als Kommunikationsschnittstelle vorgegeben. *TECS* ist ein Akronym für *Thrift Eventbased Communication System* und wurde vom Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) entwickelt [27]. Des

[27] `https://tecs.dfki.de/`

Weiteren soll der TEDURU-Server als Rahmenwerk benutzt werden können und so die Entwicklung von Mensch-Roboter Kollaborationsszenarien erleichtern und beschleunigen. In den folgenden Kapiteln wird der TEDURU-Server im Detail beschrieben.

2.3.1 Zugrundeliegende Systeme im Vergleich

Die unterschiedlichen Rahmenwerke und Systeme wurden in Bezug auf die folgenden Eigenschaften hin untersucht:

- ***Entwickelt als***

  Da es sich bei den hier angeführten Systemen, im Sinne dieses Vergleiches um Rahmenwerke handelt, auf denen Arbeiten aufgebaut werden, ist es sinnvoll die ursprünglich angedachten Ziele dieser grundlegenden Rahmenwerke zu vergleichen. Hierbei kann sich ein System um eine Soziale Kommunikations- und Interaktionsplattform handeln, welche dem Zweck der Selbstverwirklichung und dem Austausch mit anderen Spielern, bzw. Teilnehmern dient. Ein System kann auch als Serverplattform für virtuelle Welten dienen, welche singulär als Umgebung, oder auch im Verbund mit anderen als gemeinsam aufgespannte Welt, existieren kann.

- ***Persistente Welt***

  Als persistente Welt wird eine virtuelle Welt bezeichnet, welche in dem Sinne an die reale Welt angelehnt ist, sodass Veränderungen in dieser virtuellen Welt, dauerhaft für alle Nutzer, zu jedem Zeitpunkt zugänglich sind.

- ***Unterstützte Programmiersprachen***

  Systeme die es dem Anwender ermöglichen eigene Inhalte zu erstellen und diese auch mit Eigenschaften und Funktionen zu versehen, binden den Nutzer gleichzeitig auch an Restriktionen. Diese Restriktionen entstehen, da die Entwickler eines Rahmenwerkes nicht alle Möglichkeiten, Paradigmen oder Formate unterstützen können. Diese Rahmenwerke können Programmiersprachen unterstützen, welche eigens für den Gebrauch innerhalb des Rahmenwerks entwickelt wurden und welche den Nutzern zur Verfügung gestellt werden, wie *Linden Scripting Language* (LSL) oder *Open Simulator Scripting Language* (OSSL). Ein anderer Ansatz ist es den Nutzern zu ermöglichen eine bereits existierende Programmiersprache zu nutzen, um den eigens erschaffenen Inhalten Eigenschaften und Funktionalität zuzuweisen.

- ***Unterstützte 3D-Formate***

  Um dreidimensionale Modelle nutzen zu können, muss ein Rahmenwerk, entweder einen eigenen Editor für dreidimensionale Objekte zur Verfügung stellen, oder Formate unterstützen, welche von externen Programmen benutzt werden, um erstellte Modelle abzuspeichern. Ein einfacher eingebauter Editor kann zum Beispiel grundlegende Formen, sogenannte *Primitives* zur Verfügung stellen, aus denen Nutzer dann über Transformations und Translationsmethoden eigene Modelle erstellen können. Virtuelle Objekte können auf unterschiedliche Arten definiert werden. Ein Ansatz ist das Erstellen einer UV-Map die ein vorhandenes virtuelles Objekt verformt. Eine UV-Map ist ein Bild, welches über die Intensität seiner rot, grün und blau Werte den Abstand der Oberfläche zum Ursprung des Objektes bestimmt. Solch ein Objekt wird als *Sculpted Object*

bezeichnet und eignet sich um Objekte mit natürlichen Rundungen darzustellen. Ein anderer Ansatz ist das Erstellen eines *Mesh* (Polygonnetz) Objektes. Die meisten Modellierungsprogramme für dreidimensionale virtuelle Objekte arbeiten mit- und unterstützen Polygonnetzobjekte. Ein Polygonnetz ist ein Verbund aus *Vertices* (Punkten), *Edges* (Kanten) und *Faces* (Flächen). Weit verbreitete Formate für Polygonnetzobjekte sind unter anderem .fbx, .dae (Collada), .3ds, .dxf, .obj, .skp.

- ***Unterstützte Zielplattformen***

  Es existieren viele verschiedene Betriebssysteme und Plattformen, auf denen Programme laufen können. Es ist somit von Vorteil für ein Rahmenwerk, wenn mit ihm erstellte Anwendungen auf möglichst vielen Betriebssystemen laufen können.

In Tabelle 1 werden die vorher erläuterten Systeme, entsprechend einsortiert. Jedoch ist es bei einem tabellarischen Vergleich, wie er hier angewendet wird, nicht zweckdienlich, Programmiersprachen wie C# oder Bibliotheken wie Java 3D als Rahmenwerk miteinzubeziehen. Der in dieser Arbeit entwickelte TEDURU-Server basiert auf dem Rahmenwerk Unity3D. Unity3D wurde als Entwicklungsplattform für digitale Anwendungen konzipiert und entwickelt. Für entsprechend ausgelegte Unity3D Anwendungen kann eine persistente Welt über die Bereitstellung von externen Servern angeboten werden. Entwickler können im Unity3D Editor über C# Skripte, Eigenschaften und Funktionen zu vorher importierten, oder aus *Primitives* erstellten, virtuellen Objekten hinzufügen. Da Unity3D entwickelt wurde, um digitale Anwendungen zu erschaffen, unterstützt es auch eine Vielzahl an Betriebssystemen, auf denen fertige Anwendungen portiert werden können.

	Entwickelt als	Persistente Welt	Unterstützte Programmier-sprache	Unterstützte 3D-Formate	Unterstützte Zielplattformen
Second Life	Kommunikations und Interaktions-plattform	ja	LSL	*Primitives*, *Sculpted Objects*, COLLADA (.dae)	Windows, Mac, Linux
OpenSimulator	Serverplattform für virtuelle Welten	beides *	LSL, OSSL	*Primitives*, *Sculpted Objects*, COLLADA (.dae), .oar	Windows, Mac, Linux
Unity3D	Entwicklungs-plattform für digitale Anwendungen	nein +	C# Microsoft .NET, (JavaScript und Boo seit Unity3D version 2017.1 veraltet)	*Primitives*, .fbx, .dae (Collada), .3ds, .dxf, .obj, .skp	iOS, Android, Tizen, Windows, Universal Windows Platform, Mac, Linux/Steam OS, WebGL, PlayStation 4, PlayStation Vita, Xbox One, Wii U, Nintendo 3DS, Oculus Rift, Google Cardboard Android & iOS, Steam VR PC & Mac, Playstation VR, Gear VR, Windows Mixed Reality, Daydream, Android TV, Samsung SMART TV, tvOS, Nintendo Switch, Fire OS, Facebook Gameroom, Apple ARKit, Google ARCore, Vuforia

Tabelle 1: Vergleich der Systeme die als Rahmenwerke in den verwandten Arbeiten benutzt werden

* *OpenSimulator* verfügt sowohl über eine persistente Serverübergreifende virtuelle Welt, als auch über einen Modus in dem auf einen lokalen, einzelnen Server zugegriffen wird.
+ Eine Persistente Welt kann simuliert werden, indem die entsprechende Unity3D Anwendung über den entsprechenden Zeitraum nicht beendet wird, bzw. die Unity3D Anwendung auf einen externen Server zugreift, welcher Änderungen an der virtuellen Welt speichert

2.3.2 Verwandte Systeme im Vergleich

In den vorherigen Sektionen dieses Kapitels wurden verschiedene Herangehensweisen an eine Dual Reality Anwendung, deren Kommunikationsmethoden und Möglichkeiten, aufgelistet. Jedes der vorgestellten Systeme wurde mit anderen speziellen Anforderungen, an das fertige Produkt, entwickelt, weshalb auch jedes dieser Systeme seine Vorzüge in dem entsprechenden Anwendungsbereich besitzt. Ebenso ist auch der TEDURU-Server, welcher im Rahmen dieser Arbeit entwickelt wurde, für einen speziellen Anwendungsfall in MRK Szenario entwickelt worden, unter Berücksichtigung der Dual Reality Methodik.

- ***Funktionsspektrum***

  Verschiedene Systeme können trotz ähnlicher Grundkonzept, oder verwandten Anwendungsbereichen, unterschiedliche Zielsetzungen und somit auch unterschiedliche Funktionalitäten bieten. So kann das Ziel einer Dual Reality Anwendung die reine Visualisierung und somit Vermittlung von Sensorinformationen in Form eines Morphismus, zwischen der echten und der virtuellen Welt, sein. Eine andere Herangehensweise zum Dual Reality Konzept ist, die virtuelle Darstellung der physischen Welt in der virtuellen Welt als Gelände für Simulationen und Tests zu benutzen. Nutzern kann weiterhin durch Monitoring der physischen Welt ermöglicht werden die virtuelle Applikation als Leitstand zu benutzen. Schließlich kann ein System durch das Abspeichern von Nutzer- und Sensorinformationen auch zum Aufzeichnen genutzt werden.

- ***Visualisierun.***

  Es gibt mehrere Systeme die genutzt werden können, um eine echte physische Umgebung in der virtuellen Welt dreidimensional darzustellen. Hierzu zählen unter anderem *Second Life* (SL), *OpenSimulator* (OpenSim), Multiverse, sowie spezialisierte Editoren, wie zum Beispiel Unity3D.

- ***Kommunikation***

  Entsprechend zur Visualisierung gibt es auch unterschiedliche Schnittstellen, welche für einen einfachen Datenaustausch genutzt werden können. Hierzu zählen die Linden Scripting Language (LSL), vorhandene Möglichkeiten in Programmiersprachen wie zum Beispiel Phyton und eigens konzipierte Module, wie das Event Broadcasting Service (EBS) oder das *network-toolkit TECS*.

- ***Modell***

  Es gibt unterschiedliche Herangehensweise die Visualisierung einer virtuellen Welt umzusetzen. Normalerweise wird ein Modell einmalig erstellt und als Szene abgespeichert.

- ***Überwachung***

  Eine virtuelle Umgebung ist genau dann zum Monitoring einer realen Umgebung geeignet, wenn die virtuelle Umgebung möglichst realgetreu zu ihrem physischen Gegenstück ist. Hierzu bieten die entsprechenden Systeme Kommunikationsschnittstellen, um mit Information aus der echten Welt die virtuelle Welt zu beeinflussen und zu verändern.

- ***Simulation***

  Virtuelle Welten haben ein erhöhtes Potenzial als Simulationsgrundlage genutzt zu werden, um Szenarien oder Aktionen zu testen und zu evaluieren.

In Tabelle 2 werden die vorher erläuterten Systeme, entsprechend einsortiert. Der TECS based Dual Reality Unity Server (TEDURU-Server) nutzt die Netzwerk Schnittstelle des *network-toolkit TECS* als Kommunikationsschnittstelle. TEDURU wurde hauptsächlich entwickelt um ein schnelles Erstellen von Prototypen in Mensch Roboter Kollaborations Szenarien (MRK-Szenarien), unter zuhilfenahme von Dual Reaity Methoden, zu ermöglichen. Hierfür nutzt der TEDURU-Server Unity3D als Grundlage, da Unity3D aufgrund seines Ursprungs als Entwicklungsplattform für dreidimensionale Spiele, über ausführliche Möglichkeiten und Fähigkeiten zur Visualisierung und Manipulation von virtuellen dreidimensionalen Objekten und Umgebungen verfügt. Über das Abspeichern des Zustands der virtuellen Welt können Vorgänge überwacht und rekonstruiert werden. Gleichzeitig kann die virtuelle Welt des TEDURU-Servers benutzt werden um Szenarien zu simulieren und zu evaluieren.

	Funktions-spektrum	Visualisierung	Kommuni-kation	Modell	Überwachung	Simulation
Kontextfelder	Morphismus	SL	LSL	Szene	ja	nein
Twin-World Mediator	Testumgebung	SL/OpenSim	Twin-World Mediator/LSL	Szene	nein	ja
Eolus One	Leitstand	SL/OpneSim	LSL	Szene	ja	nein
Virtuelle Schockoladenfabrik	Leitstand	Multiverse (später Unity3D)	Python Skripte	Szene	ja	nein
YAMAMOTO	Leitstand/ Testumgebung	YAMAMOTO (C#)/ VRML/ RoSiML-Viewer	Rest/VNC/UCI	Szene (yml)	ja	*
DuRMaD + EBS	Leitstand/ Testumgebung/ Aufzeichnung	Java3D/XML3D	Rest/VNC/EBS	dynamisch (yml/gml)	ja	ja
TEDURU-Server + TECS	Leitstand/ Testumgebung/ Aufzeichnung	Unity3D	*network-toolkit TECS*	dynamisch	ja	ja

Tabelle 2: Vergleich der verwandten Dual Reality Arbeiten mit dem TEDURU-Server

* Enthält eine Abspielfunktion von virtuellen Avataren, sonst keine Simulation

3 Konzept und Implementierung

Ein *Framework* (Rahmenwerk) muss nicht nur ein System implementieren, oder ein Szenario umsetzen, sondern darüber hinaus auch die Möglichkeit bieten, für eine große Auswahl an Systemen und Szenarien als Grundlage zu dienen und somit deren Ansprüchen entsprechen. Die geforderten Ansprüche an das Rahmenwerk, das in dieser Arbeit beschrieben wird, sind zum einen, dass die Umsetzung und Fertigung von MRK-Szenarien mit dem Rahmenwerk nicht nur beschleunigt, sondern auch vereinfacht wird. Zum anderen soll das Rahmenwerk auf einem Kommunikationsprotokoll basieren, welches für mehrere MRK-Szenarien wiederverwendet werden kann. Um solch ein Rahmenwerk implementieren zu können, sind viele teils kleinere und auch unabhängige, elementare Konzepte nötig, die zusammen ein vielfältig nutzbares System bilden, welches den Anforderungen entspricht. Aus diesem Grund wird in diesem Kapitel, Konzept und Implementierung, direkt aufeinander folgend, zu gleichen Teilen behandelt, anstatt in aufeinanderfolgenden Kapiteln nacheinander bearbeitet.

3.1 Kommunikation und Rahmenwerk

Ein Rahmenwerk zum schnellen Entwickeln von Prototypen für MRK Szenarien erfordert eine flexible Kommunikationsstruktur, die es ermöglicht, zwischen verschiedensten Geräten innerhalb einer Smarten Umgebung zu kommunizieren. So muss es zum Beispiel möglich sein über die Positionsdaten einer Hololens, unterschiedliche Events auf einer Smartwatch zu aktivieren, während Informationen von einem mobilen Roboter empfangen, verarbeitet und wiederum an alle Beteiligten weitergeleitet werden.

3.1.1 Konzept

Die Kommunikation zwischen unterschiedlichen Robotern, Sensoren und Systemen ist eine Grundlage von MRK-Szenarien, denn ohne Kommunikation der einzelnen Komponenten, kann keine vernünftige Zusammenarbeit zwischen Mensch und Maschinen funktionieren. Da eine Entwicklung einer neuen Kommunikationsinfrastruktur zusätzlich zur Entwicklung des TEDURU-Servers, den Rahmen einer Bachelorarbeit überschreiten würde, wurde die Verwendung einer schon existierenden Kommunikationsinfrastruktur samt eines Rahmenwerkes, die diese Kommunikationsinfrastruktur unterstützt, vorgegeben.

3.1.2 Implementierung

Aufgrund der weitläufigen Unterstützung vom *network-toolkit TECS* für verschiedenste Programmiersprachen und der Systemunabhängigkeit, die es bietet, sowie bestehenden Erfahrungen mit dem *network-toolkit TECS*, wurde entschieden, das *network-toolkit TECS* auch für die Kommunikationsinfrastruktur des TEDURU-Servers zu verwenden. Das *network-toolkit TECS* ist eine auf thrift [47] basierende Kommunikationsstruktur, welche Systeme unabhängig benutzt werden kann, weshalb es kaum Beschränkungen in der Auswahl des Rahmenwerkes welches verwendet werden soll, gibt. In Betracht gezogen wurden unterschiedliche System, die auch von anderen Entwicklern benutzt wurden und sich bewiesen haben. So gibt es Rahmenwerke zum Entwickeln von Spielen und digitalen Anwendungen wie Unreal [28] [10, 15], oder

[28] `https://www.unrealengine.com/en-US/what-is-unreal-engine-4`

Unity3D [29] [24, 46, 14, 16]. Robotersimulationen wie Webots [30] [23, 36], oder eine Kombination aus einer weit verbreiteten Stage [31] und Gazebo [32] Middleware [45, 28, 22]. Sogar eine rein mathematische Herangehensweise über Matlab [33] [19, 48, 44] wurde in Betracht gezogen. Jedes dieser Rahmenwerke hat seine eigenen Vor- und Nachteile und die Wahl eines geeigneten Rahmenwerkes ist eine projektspezifische Entscheidung, die in Einklang mit den Anforderungen des Projekts und der Expertise der Entwickler abgeglichen werden muss.

Da die Anforderungen eine visuelle Präsentation der virtuellen Umgebung vorsehen und entsprechende Erfahrungen mit Unity3D vorhanden sind, wurde Unity3D als Rahmenwerk für den TEDURU-Server benutzt. Zum Nachteil wurde hierbei jedoch, dass Unity3D nicht *threadsicher* konzipiert ist. Somit ist es zusätzlichen Threads in einem Unity3D Programm nicht möglich, auf wichtige Unity3D Funktionalitäten zuzugreifen. Da für einen *Remote Procedure Call* (RPC) mit thrift ein eigener Thread benötigt wird, übersteigt die Implementierung eines *network-toolkit TECS*-RPC-Unity3D Servers den Rahmen dieser Bachelorarbeit, weshalb auf eine *Publish-Subscribe* (PS) Kommunikation ausgewichen wurde.

3.2 Visuelles Debugging

Im Laufe dieser Arbeit wird visuelles Debugging nicht als eine besondere Form eines Debuggers für geschriebenen Code verstanden, sondern als eine visuelle Darstellung einer virtuellen Szene, welche eine reale Umgebung, samt Sensoren und Agenten, widerspiegelt. Hierbei soll die virtuelle Darstellung alle nötigen Informationen visuell zur Verfügung stellen, die benötigt werden, um auf einen Blick erkennen zu können, ob alle in der realen Umgebung vorhandenen Sensoren, Aktuatoren und Agenten in dem zusammengeführten, gemeinsamen, virtuellen Weltbild, an der jeweils richtigen Position, Rotation und Skalierung, sind.

3.2.1 Konzept

In einem Mixed Reality Szenario ist die Fähigkeit des visuellen Debuggens eine Grundvoraussetzung um Änderungen schnell auswerten zu können und grobe Fehlerquellen schneller zu finden. Dies beruht darauf, dass das Zusammenbringen der echten und der virtuellen Welt keine triviale Aufgabe ist. So hat meist jeder Agent in einem MRK-Szenario, sein eigenes Koordinatensystem in seinem eigenen Weltmodell. Ein zentraler Server mit einer dreidimensionalen Darstellung der Szene erleichtert hierbei das Zusammenbringen aller Sensoren, Aktuatoren und Agenten, da so jederzeit die Position jeder Komponente auf einen Blick überprüft werden kann, wie in Abbildung 19 zu sehen ist.

Hierzu ist es nicht nur notwendig, dass alle wichtigen Komponenten der echten Umgebung dargestellt werden können, sondern auch, dass der Nutzer des Systems die Dargestellte virtuelle Szene in ausreichendem Maße betrachten kann, ihm also keine notwendigen Informationen vorenthalten bleiben.

[29] `https://unity3d.com/`
[30] `https://www.cyberbotics.com/`
[31] `http://playerstage.sourceforge.net/index.php?src=stage`
[32] `http://gazebosim.org/`
[33] `https://de.mathworks.com/products/matlab.html`

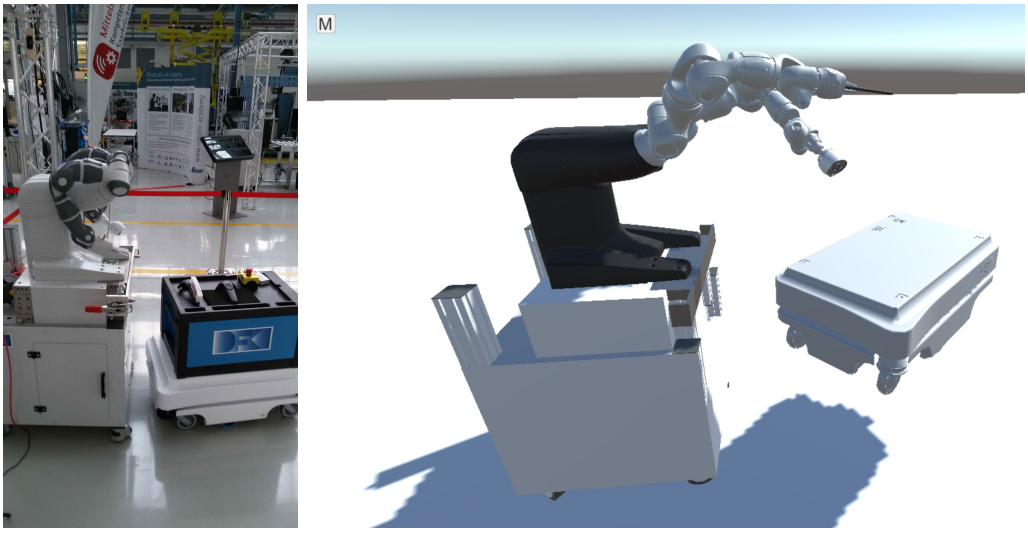


Abbildung 19: *Vergleich zwischen einer Szene in der echten Welt (links) und ihrer fehlerhaften virtuellen Repräsentation (rechts), zu erkennen an fliegender Mir100 Plattform mit falscher Orientierung.*

3.2.2 Implementierung

Die Umsetzung einer ausreichenden visuellen Darstellung für den Zweck des visuellen Debuggens wurde durch die Verwendung von Unity3D als Rahmenwerks für den TEDURU-Server erfüllt, da Unity3D als modernes 3D-Rahmenwerk schon ausreichende Visualisierungsmöglichkeiten für dreidimensionale Modelle liefert, wie in Abbildung 20 zu sehen ist.

Weiterhin ist die richtige Positionierung einer virtuellen Kamera, durch welche die virtuelle Umgebung gesehen wird, ein eigenes Forschungsfeld und von Szenario zu Szenario unterschiedlich [37]. Deshalb wurde eine frei bewegliche und frei rotierbare virtuelle Kameraführung implementiert, damit ein Nutzer die virtuelle Kamera zu jedem Zeitpunkt zu einer optimalen Position, in der virtuellen Umgebung, bewegen kann. Die virtuelle Kamera hat für diese Zwecke keinerlei Kollision, wodurch sie durch andere virtuelle Objekte hindurchbewegt werden kann und somit auch *in* Robotern oder Objekten positioniert werden kann, ohne Kollisionen oder andere Ereignisse auszulösen. Bewegungsmöglichkeiten für die Kamera umfassen eine lineare dreidimensionale Bewegung auf den eigenen lokalen Achsen (X,Y,Z), sowie Rotation um die X und Z Achse. Rotationen, um die Y Achse wurde aufgrund von möglichen Orientierungsproblemen durch *schräge* Kameraführung ausgelassen. Eine lineare Bewegung kann gleichzeitig auch mit einer Rotation der Kamera kombiniert werden, um eine flüssigere Navigation durch den virtuellen Raum zu ermöglichen. Schlussendlich bietet der TEDURU-Server auch noch eine minimalistische UI. Daraus ergibt sich sowohl die Möglichkeit, die Kamera wieder in die Ausgangsposition zu setzen, als auch die Möglichkeit, die Kamera auf ein gewünschtes Objekt zu zentrieren. Dadurch kann die Suche nach bestimmten Objekten in der Umgebung vereinfacht werden.

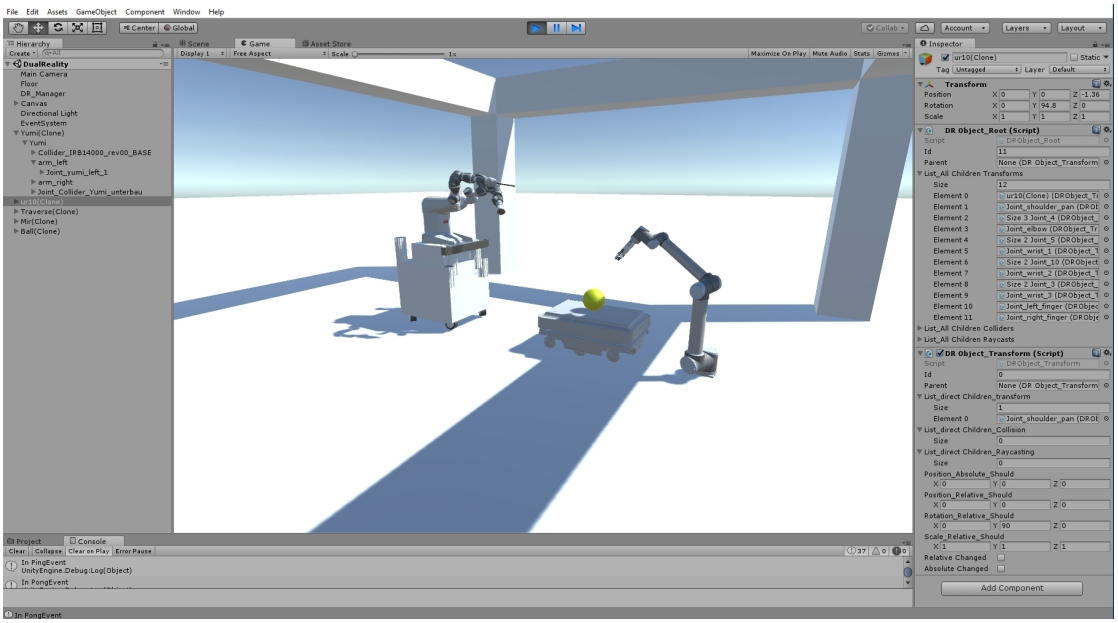


Abbildung 20: *Laufender TEDURU-Server in Unity3D Umgebung*

3.3 Automatische Integration von Modellen

Mit automatischer Integration von Modellen ist in diesem Unterkapitel gemeint, dass ein vom TEDURU-Server geladenes Modell, ohne manuelles Nachbessern des Nutzers, automatisch benutzt werden kann. Eine vorherige externe Zuweisung von Skripten soll durch die automatische Integration von Modellen des TEDURU-Servers auch vermieden werden. Nicht gemeint ist, dass zum Beispiel alle virtuellen Modelle auf einem Rechner, oder in einem speziellen Ordner automatisch geladen werden.

3.3.1 Konzept

Mit dem TEDURU-Server wird ein selbstständig lauffähiges abgeschlossenes System geliefert, welches als eine Art Blackbox benutzt werden kann, wodurch Nutzer des Systems nicht gezwungen sind das System zu studieren und zu lernen. Die Modelle, die im TEDURU-Server benutzt werden, werden jedoch, wie für dreidimensionale virtuelle Objekte üblich, mit spezialisierten Modellierungswerkzeugen wie zum Beispiel Blender [34], Autodesk Maya [35], oder auch AutodeskCAD [36], für CAD Modelle, erstellt. Diese extern erstellten Modelle müssen nun vom TEDURU-Server benutzt werden können. Da das in dieser Arbeit entwickelte System möglichst einfach zu nutzen sein soll, sollte es auch für einen Laien der 3D Modellierung möglich sein ein vorhandenes virtuelles Modell für den Gebrauch mit dem TEDURU-Server nutzbar zu machen. Um den TEDURU-Server als Blackbox zu halten und eine einfache Integration von neuen virtuellen Modellen zu bieten, sollte die Integration von neuen geladenen Modellen somit möglichst automatisch geschehen, ohne dass TEDURU-Skripte, welche Funktionen

[34] `https://www.blender.org/`
[35] `https://www.autodesk.de/products/maya/overview`
[36] `https://www.autodesk.de/products/all-autocad`

und Eigenschaften enthalten, dem entsprechenden Modell oder seinen Untermodellen manuell zugewiesen werden müssen.

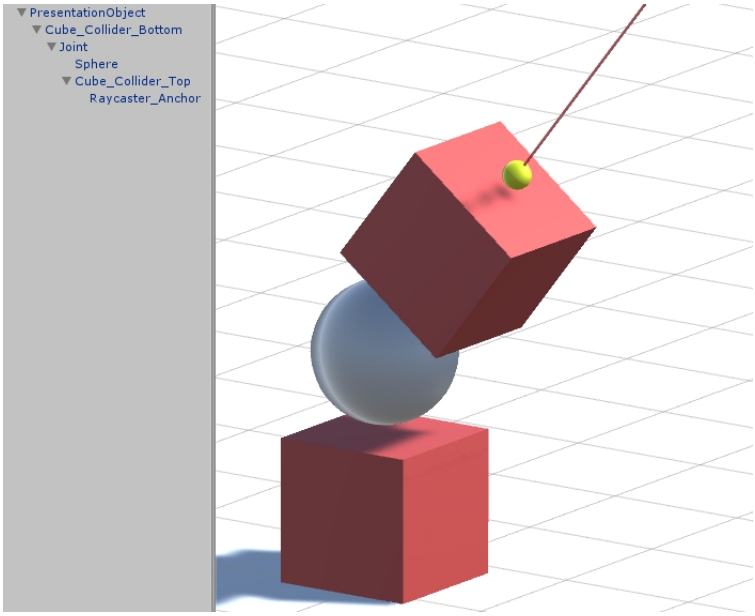


Abbildung 21: *Baumstruktur einer simplen Repräsentation eines ein-gelenkigen Greifarms in Unity3D*

Ein weiterer Aspekt, welcher zu der Integration von virtuellen Modellen in TEDURU gehört, ist der Umgang mit den Modellen. Es ist nicht erforderlich, dass virtuelle Objekte lediglich aus einem einzelnen Modell bestehen. So hat zum Beispiel das virtuelle Objekt eines Greifarms, genau wie sein physisches Gegenüber, mehrere Gelenke, welche allesamt bewegt werden müssen, damit sich der virtuelle Greifarm genau wie der echte Greifarm bewegen kann. Solch eine Baumstruktur wird auch in Abbildung 21 anhand einer vereinfachten Darstellung eines ein-gelenkigen Greifarms dargestellt. Die einzelnen Funktionen der Unterkomponenten, wie Gelenke, die sich drehen können, oder auch feste Objekte, die kollidieren können, müssen einzeln ansprechbar sein.

3.3.2 Implementierung

Der TEDURU-Server nutzt bei der Integration von virtuellen Objekten, deren modularen, baumstrukturartigen Aufbau. Wenn ein Modell geladen wird, iteriert der TEDURU-Server durch die Baumstruktur und sucht in den Bezeichnungen der Untermodelle nach vordefinierten Stichworten. Wenn solch ein Stichwort gefunden wurde, werden automatisch alle dem Stichwort zugeordneten Skripte zu dem Untermodell hinzugefügt. Diese Stichworte sind:

- ***Joint*** 3.5

  gibt dem Objekt Möglichkeiten über Transformationsevents in seiner Position, Rotation und Skalierung verändert zu werden.

- ***Collider*** 3.7

  erstellt eine unsichtbare Kollisionsbox, die das Objekt einschließt und Kollisionen mit Kollisionsboxen anderer Objekte erkennen kann.

- ***Raycaster*** 3.8

  fügt dem Objekt ein Raycast-Objekt hinzu, welches kontinuierlich einen Strahl aussendet. Wenn dieser Strahl auf eine Kollisionsbox trifft, gibt das Raycast-Objekt das Objekt, zu dem die Kollisionsbox gehört, sowie die Entfernung zu dem Treffer zurück. Dies kann zur Erkennung von Blickrichtung oder Zeigegesten genutzt werden.

- ***Anchor*** 3.9

  erstellt ein Ankerpunktobjekt an der Position des annotierten Objektes. Dieser Ankerpunkt kann nun zum Beispiel durch ein *MoveTo* Event angesprochen werden, um ein anderes Objekt zu diesem Ankerpunkt bewegen zu können.

Auf die einzelnen Komponenten wird in den nachfolgenden Kapitel noch genauer eingegangen.

Zusätzlich zu den vier oben genannten Komponenten, gibt es noch die *Root* (Wurzel) Komponente, welche jedem virtuellen Objekt, an die Wurzel des virtuellen Modellbaums hinzugefügt wird. Diese Wurzel wird intern verwendet, um alle in diesem Objekt vorhandenen Komponenten abzuspeichern und somit auch um auf diese zuzugreifen.

Um die einzelnen Komponenten eines virtuellen Objektes ansprechen zu können, wurden mehrere Herangehensweisen in Betracht gezogen und gegeneinander abgewogen. Zur Vereinfachung und Verdeutlichung dieses Abschnittes und auch der folgende Abschnitte werden die Begriffe *Parent*, *Child* und *Komponente* entsprechend ihres Gebrauchs im Zusammenhang mit dem TEDURU-Server definiert.

Als *Parent* wird ein vollständiges Modell als Ganzes bezeichnet. So besteht ein virtuelles Objekt, wie oben beschrieben, meist aus mehreren kleineren Objekten, welche ihrerseits nochmal aus weiteren Objekten bestehen können.

Ein *Child* ist entsprechend eines der Unterobjekte zu einem *Parent*. Ein *Parent* hat somit viele *Children*, welche ihrerseits wiederum weitere *Children* haben können.

Als Komponente wird ein virtuelles Objekt im TEDURU-Server bezeichnet, welches mit einem der oben aufgelisteten Stichwörter versehen wurde und dadurch direkt ansprechbar ist. Somit kann über eine Komponente auf die Fähigkeiten eines Objektes, wie zum Beispiel eine Veränderung der Position oder das Abfragen von aktuellen Kollisionen, zugegriffen werden. Hierbei ist zu beachten, dass jede Komponente ein *Child* ist, aber nicht jedes *Child* eine Komponente.

Als erste Herangehensweise wurde jeder Komponente eine eigene ID zugewiesen. Vorteile dieser Herangehensweise sind, dass jede Komponente eindeutig und ohne Berechnungen über eine ID bestimmt werden kann und dass keine maximale Anzahl an untergeordneten Komponenten für ein *Parent* existiert, wie es bei der nächsten Methode der Fall ist. Nachteil ist, dass anhand der ID nicht ersichtlich ist, welche Art von Objekt angesprochen wird.

Die zweite Herangehensweise ist eine zusammengesetzte ID für jede Komponente. Hierbei bekommt jedes *Parent* eine eigene einzigartige fortlaufend ID, angefangen bei 1. Jede Komponente eines *Parent* bekommt auch eine eigene einzigartige fortlaufend ID, welche auch hier bei 1 anfängt. Die gesamte ID, zum ansprechen einer Komponente, ergibt sich nun durch das aneinanderhängen der beiden IDs, wobei erst die *Parent*-ID kommt und dann

die Komponenten-ID, welche durch Nullen auf drei Ziffern erweitert wird. Um nun zum Beispiel die zwölfte Komponente eines *Parents*, welches die ID 23 besitzt, anzusprechen, müsste die ID 23012 mitgegeben werden. Vorteile an diesem System ist, dass es einfach ist die ID einer Komponente zu bestimmen. Nachteil ist, dass eine Maximalanzahl an möglichen IDs für Komponenten in jedem *Parent* existieren, da sonst die ID der Komponente die ID des *Parent* beeinflusst, wodurch die gesamte ID verfälscht wird.

Die letzte Herangehensweise ist eine Doppel ID Herangehensweise. Die IDs werden genauso wie bei der zweiten Herangehensweise verteilt. Der Unterschied zur zweiten Herangehensweise ist jedoch, dass keine zusammengesteckte ID gebildet wird, sondern, dass jede Komponente über zwei IDs angesprochen wird. Entsprechend dem Beispiel des zweiten Ansatzes wird die Komponente nicht über die ID 3012, sondern über die *Parent*-ID 3 und die *Child*-ID 12 angesprochen. Vorteil ist hier, dass es kein Maximum an möglichen *Child*-IDs gibt, während auch eindeutig ersichtlich ist welches *Child* von welchem *Parent* angesprochen wird. Nachteil ist, dass zwei IDs anstatt, wie bei den vorherigen Herangehensweisen, nur eine übermittelt werden müssen. Da die IDs jedoch als Bestandteil eines Events versendet werden, sollte der nötige Mehraufwand einer *Child*-ID jedoch vernachlässigbar sein. Implementiert wurde diese dritte Herangehensweise.

Zusätzlich besitzen Objekte im TEDURU-Server einen Modellnamen und einen Eigennamen. Der Modellname entspricht dem Namen über den im TEDURU-Server bestimmt wird was für ein Objekt initialisiert wird. Der Modellname muss einzigartig sein und ist unveränderlich. Der Eigenname hingegen kann einem Objekt beim Initialisieren beliebig gegeben werden, selbst wenn solch ein Eigenname schon vergeben wurde. Er dient einzig dem Nutzer als mögliches Hilfsmittel zur Gliederung von Szenen, da er weder zur Registrierung noch für sonstige Anfragen genutzt wird.

Im Folgenden ist der Quellcode der thrift Klasse, eines vollständigen TEDURU-Objekten zu sehen. Die *Parent*-ID wurde in der Implementierung *RootID* genannt. Zeilen die mit // anfangen, sind Kommentare.

```
1  struct TEDURUObject {
2      // REQUIRED
3      1: required i64 RootID;
4      2: required i64 ChildID;
5      3: required string modelName;
6
7      // Optional but mostly there
8      4: optional string objectName;
9      5: optional list<string> tags
10     6: optional General3D.Vector3d absolutePosition;
11     7: optional General3D.Vector3d relativePosition;
12     8: optional General3D.Vector3d rotation;
13     9: optional General3D.Vector3d scale;
14
15     // Optional
16     10: optional TEDURUObject parent;
17     11: optional list<TEDURUObject> children;
18     12: optional double distanceThisObjectHasToTheRaycastOrigin;
19 }
```

3.4 Laden von Modellen

Der normale Arbeitsweg ein Modell in einer Unity3D Anwendung zu laden, beinhaltet das Einbinden des Modells in das Projekt, bevor dieses gebaut wird. Das Laden eines externen Modells während der Laufzeit wird standardmäßig nicht unterstützt.

3.4.1 Konzept

Eine weitere Anforderung an den TEDURU-Server ist die Möglichkeit virtuelle Modelle in den Server zu integrieren, ohne ihn dafür ausschalten, oder seine aktuelle Ausführung unterbrechen zu müssen.

3.4.2 Implementierung

Unity3D unterstützt viele verschiedene und weit verbreitete Modellformate wie .fbx, .dae, .3ds, .dxf, .obj, und .skp [37]. Jedoch können die entsprechenden Modelle nur von Unity3D geladen werden, wenn gleichzeitig auch ein Modellierungsprogramm installiert ist, welches das entsprechende Format öffnen kann. Außerdem erlaubt Unity3D standardmäßig keinen Zugriff auf die interne Ordnerstruktur eines fertig gebauten Projekts. Durch die Nutzung des *StreamingAssets* [38] Ordner, auf welchen sowohl von außerhalb, als auch von innerhalb einer fertig gebauten Unity3D Anwendung zugegriffen werden kann und *AssetBundles* [39] ist es jedoch möglich, Modelle in ein fertig gebautes Unity3D Projekt, während der Ausführung, zu laden. *AssetBundles* ist ein Unity3D spezifisches Dateiformat. Der Unity3D-Editor kann jedes geladene Objekt von den unterstützten Modellformaten in ein *AssetBundles* umwandeln. Durch die Kombination des *StreamingAssets* Ordner und der *AssetBundles* ist es somit möglich, Modelle auch während einer laufenden Unity3D Anwendung in das Laufende Projekt zu laden. Über die Funktion der automatischen Integration 3.3 können diese geladenen Modelle auch den vollen Umfang der TEDURU-Server Funktionalität nutzen.

3.5 Transformationen

Die Bewegung von realen, sowie virtuellen Objekten, wird als Transformation bezeichnet. Die Möglichkeit virtuelle Objekte transformieren zu können ist somit eine notwendige Grundlage eines jeden Systems mit nicht statischer dreidimensionaler Darstellung.

3.5.1 Konzept

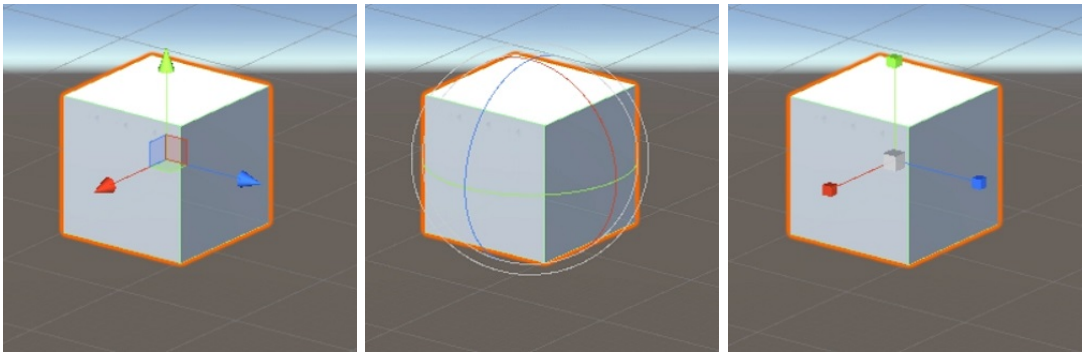
MRK Anwendungen sind per Definition des Begriffes [40] keine Szenarien in denen sich Roboter statisch verhalten, da statische unbewegliche Roboter nicht kollaborieren, sondern eher als Hindernis fungieren. Um nun die virtuelle Darstellung den realen Begebenheiten nachzuempfinden, müssen die virtuellen Modelle bewegt, also transformiert werden können. Darüber hinaus bestehen virtuelle Objekte, wie im vorherigen Abschnitt beschrieben, nicht nur aus einem einzelnen Modell, sondern aus vielen Unterobjekten, welche in einer Baumstruktur angeordnet sind. Da diese Struktur Auswirkungen auf die Transformation von Objekten hat, muss sie deshalb auch beachtet werden.

[37] `https://docs.unity3d.com/Manual/3D-formats.html`
[38] `https://docs.unity3d.com/Manual/StreamingAssets.html`
[39] `https://docs.unity3d.com/Manual/AssetBundlesIntro.html`
[40] `https://de.wikipedia.org/wiki/Kollaboration`

3.5.2 Implementierung

Die Grundlagen die abgedeckt werden müssen, um eine freie Beweglichkeit im virtuellen Raum zu ermöglichen, sind die sechs Freiheitsgrade [41] die ein normaler Gegenstand in der echten Welt auch besitzt. Im virtuellen Raum kommen noch drei weitere Freiheitsgrade hinzu, über welche ein Objekt in seiner Größe verändert werden kann. Hierbei ist zu beachten, dass virtuelle Objekte gleichzeitig Bestandteil von zwei unterschiedlichen Koordinatensystemen sein können. Virtuelle Objekte sind, genau wie die meisten Gegenstände in der echten Welt, aus unterschiedlichen Bestandteilen aufgebaut, die gemeinsam, ein ganzes Objekt bilden. Im virtuellen wird solch ein Zusammenhang, dass ein Objekt aus mehreren Unterobjekten besteht, meist durch eine Baumstruktur der einzelnen Objekte dargestellt, wie in Abbildung 21 zu sehen ist. Im Fall von Unterobjekten ist nun jedes Unterobjekt in zwei unterschiedlichen Koordinatensystemen eingeteilt. Zum einen in ein globales, oder auch absolutes Koordinatensystem, das der ganzen virtuellen Welt als Grundlage dient. Zum anderen ein lokales, oder auch relatives Koordinatensystem. Solch ein lokales Koordinatensystem eines untergeordneten Objektes hat seinen Ursprung in dem ihm übergeordneten Objekt und übernimmt auch dessen Rotation und Skalierung. Um als Beispiel das Objekt aus Abbildung 21 zu nehmen, so steht das Objekt als Ganzes im Ursprung und hat die globalen, wie auch die lokalen Koordinaten (0,0,0). Der obere Teil des Greifarms hingegen hat nun unterschiedliche globale und lokale Koordinaten. So befindet sich der zweite Arm an der globalen Position (0.68,2.94,5.96) (Rote Blöcke und Kugel haben jeweils einen Durchmesser von Eins), während er sich an der lokalen Position (0,1.11,0) befindet. In Unity3D ist außerdem, wie am Beispiel zu erkennen ist, die Y-Achse die Applikate, während die Z-Achse als Ordinatenachse mit der X-Achse den dreidimensionalen Raum aufspannt.

Ein weiterer Punkt, der bei der Implementierung von Transformationen in Unity3D zu beachten ist, ist, dass Rotationen in Unity3D als Quaternionen implementiert wurden. Im Editor werden die Werte jedoch zur einfacheren Lesbarkeit in der Eulerdarstellung angezeigt.



(a) *Positionsgreifer* (b) *Rotationsgreifer* (c) *Skalierungsgreifer*

Abbildung 22: *Greifer zum manuellen Verändern der Position (Links), Rotation (Mitte) und Größe (Rechts) eines Objektes im Unity3d-Editor.*

[41] `https://de.wikipedia.org/wiki/Freiheitsgrad`

3.5.3 Position

Zur Positionierung wurde aufgrund der oben genannten Gründen jeweils ein Event implementiert, um die globale Position und ein Event um die lokale Position eines virtuellen Objektes zu verändern. Zusätzlich wurde noch ein Event implementiert um ein Objekt an die globale Position eines anderen Objektes zu bewegen. Durch das Konzept der Ankerpunkte, welches in einem späteren Absatz erläutert wird, können so Objekte auf oder auch in bestimmte vorgegebene Positionen bewegt werden, ohne dass die exakte Position bekannt ist. Somit kann zum Beispiel ein virtueller Gegenstand in eine Kiste in einem Regal gelegt werden, ohne dass durch vorheriges Nachschauen oder Berechnungen im dreidimensionalen Raum die exakte Position der Kiste bekannt ist.

3.5.4 Rotation

Für die Rotation gibt es ein Event, welches Einfluss auf die lokale Rotation eines Objektes nehmen kann. Die Entscheidung keine Events für globale Rotationen zu implementieren stammt daher, dass virtuelle Objekte, genau wie echte Objekte, aufeinander aufbauend funktionieren. So orientiert sich auch bei Robotern ein Gelenk eines Greifarms standardmäßig am vorherigen Gelenk, was einer lokalen Rotation entspricht und nicht an einem zum restlichen Greifarm unabhängigen globalen Rotationssystem.

3.5.5 Skalierung

Für die Skalierung wurde, genau wie bei der Rotation, nur ein Event für lokale Skalierungen implementiert. Dies hat den Grund darin, dass Objekte für gewöhnlich nicht, oder nur initial skaliert werden müssen, da ihre virtuellen Modelle schon maßstabsgetreu modelliert werden sollten, bevor sie in den TEDURU-Server importiert werden. Somit dient die Skalierbarkeit von Objekten im TEDURU-Server in erster Linie nur dazu, den Maßstab von Objekten zu verändern, und nicht dazu, einzelne Bestandteile eines Objektes zu skalieren. Des Weiteren führt das Skalieren eines *Parents* dazu, dass sich alle *Children*, durch die Veränderung ihres lokalen Koordinatensystems, auch entsprechend skalieren.

Zusätzlich wurde noch ein Event implementiert, mit dem die lokale Position, Rotation und Skalierung eines Objektes gleichzeitig angepasst werden kann. Somit müssen von einem Objekt nicht mehrere Events ausgehen, um sein virtuelles Gegenstück zu aktualisieren. Sämtliche hier beschriebenen Eigenschaften sind im TEDURU-Server ausschließlich für Komponenten mit dem *Joint* Stichwort verfügbar, was diese Komponenten in eine besondere Position befördert. So sind die Gelenke die einzigen Komponenten innerhalb von Objekten die bewegt werden können, während die drei anderen Komponenten *Anchor*, *Collision* und *Raycast* nur Eigenschaften zum An- und Ausschalten sind. Zusätzlich ist es möglich den aktuellen Zustand von *Collision* und *Raycast* Komponenten abzufragen, also ob zurzeit Kollisionen stattfinden oder ob Objekte mit einem *Raycast* getroffen werden.

3.6 Speichern und Laden von Szenen

Der Zustand einer virtuellen Umgebung setzt sich beim Schließen der Anwendung im Normalfall wieder in den Ausgangszustand zurück. Um die virtuelle Umgebung nicht bei jedem Neustart neu aufbauen zu müssen, ist es sinnvoll den Zustand der Umgebung zu speichern, so dass er bei einem erneuten Start der Anwendung geladen werden kann.

3.6.1 Konzept

Der TEDURU-Server soll Funktionen zum Speichern und Laden von Szenen besitzen, sodass es nach Ausschalten und erneutem Anschalten des Servers nicht erforderlich ist, alle Komponenten erneut einzeln zu laden und, wenn nötig, diese mit zusätzlichen Informationen zu versehen.

3.6.2 Implementierung

Um eine TEDURU-Server-Szene zu speichern, müssen drei Dinge festgelegt werden. Als Erstes muss bestimmt werden, **was** für Informationen und Daten notwendig sind, um eine entsprechende Szene so zu reproduzieren, dass ein Szenario ohne Komplikationen mit ihr, sowohl vor dem Speichern, als auch nach dem Laden, funktioniert. Als Zweites muss bestimmt werden, **wie** die gesammelten Daten gespeichert werden und als Drittes muss noch bestimmt werden, **wo** die Daten abgespeichert werden. Entsprechend wird im Folgenden Abschnitt auf die drei notwendigen Schritte eingegangen.

Informationen, die trivial notwendig sind, um eine virtuelle Szene zu rekonstruieren, sind die Modelle, die in die Szene geladen wurden, sowie ihre Position, Rotation und Skalierung in der Szene. Die Modelle können hierbei einfach über ihren jeweiligen Modellnamen gespeichert werden. Weiterhin ist wichtig, dass die Modelle die gleiche ID und den gleichen Eigennamen nach dem Laden besitzen, die sie vor dem Speichern hatten. Eine weitere wichtige Information besteht darin, ob die Kollisionserkennung eines Modells an oder ausgeschaltet ist. Gleiches gilt auch für den Zustand der Raycasting Eigenschaften und ob sie für Raycasts sichtbar sind. Schlussendlich müssen noch alle vergebenen Tags und eventuelle zusätzliche Informationen 3.11 der Modelle abgespeichert werden. Der Zustand von Child-Objekten wird bei diesem Ansatz bewusst nicht mit abgespeichert, da in den meisten Fällen die Child-Objekte, die verändert werden, sich durch Benutzung verändern, wie zum Beispiel die einzelnen Gelenke eines Greifarmes. Somit befinden sich die Child-Objekte nach dem Laden in ihrer Ausgangsposition, bis sie wieder in Gebrauch genommen werden. Durch ein erneutes In Gebrauch nehmen der Umgebung, werden die Positionen der Unterobjekte auf ihren aktuellen Stand, in Bezug zu ihrem Parent gebracht. Wenn vorhanden, werden Hexagonkarten 3.10 auch abgespeichert. Um eine Hexagonkarte abzuspeichern, reichen genau die Informationen, die notwendig sind, um solch eine Karte zu erstellen, sowie die ID die der Karte zugewiesen wurde. Dies umfasst den Offset der Karte, die Maße der Karte in x, y, und z Richtung, sowie den Radius der Hexagone in der Karte.

Um alle oben aufgelisteten Informationen und Daten zu speichern gibt es mehrere Möglichkeiten. So ist es möglich Daten über eine Kombination von PHP und MySQL auf einem externen Server zu speichern und auch wieder von dort abzurufen [42]. Es ist möglich .NET/Mono Serialization [43] oder auch JSON [44] zu benutzen. Unity3D kann auch mit SQLite [45] genutzt werden. Da Unity3D das .Net Framework unterstützt, kann auch über die StreamWriter-Klasse [46], die Daten in ein eigenes Format überführt werden. Im fertigen Projekt ist eine vorhandene .Net/Mono Serialization Lösung implementiert, da diese ohne externe, oder zusätzliche Anwendungen umgesetzt werden kann.

[42] http://wiki.unity3d.com/index.php?title=Server_Side_Highscores
[43] https://docs.unity3d.com/Manual/script-Serialization.html
[44] http://wiki.unity3d.com/index.php/SimpleJSON
[45] https://answers.unity.com/questions/743400/database-sqlite-setup-for-unity.html
[46] https://msdn.microsoft.com/de-de/library/system.io.streamwriter(v=vs.110).aspx

Im Folgenden wird eine Sequenz eines Quellcodes aufgezeigt, die die Klassen beschreibt, welche die Daten mit der .Net/Mono Serialization abspeichern.

```
public class SaveData
{
    public List<SerObj> Contents = new List<SerObj>();
    public List<SerHexagonMaps> hexagonGrids = new List<SerHexagonMaps>();
    public List<string> tags;
}
```

```
public class SerHexagonMaps
{
    public long id;
    public SerVector3 gridDimensions;
    public SerVector3 gridOffset;
    public float hexagonRadius;
}
```

```
public class SerObj
{
    public string modelName;
    public string objectName;
    public List<string> tags;
    public long id;
    public SerVector3 absolutePosition;
    public SerVector3 rotation;
    public SerVector3 scale;
    public bool toggleRaycast;
    public bool toggleCollision;
    public bool isVisibleForRaycasts;
    public string additionalInfo;
}
```

Die Dateien werden in einem Unterordner des *StreamingAssets* Ordners abgelegt. Somit ergibt sich die Möglichkeit, eine gespeicherte Szenen in unterschiedlichen TEDURU-Servern zu benutzen, was unter anderem auch die Versionierung von MRK-Szenarien erleichtern kann.

3.7 Kollisionen Erkennen

Im Gegensatz zu realen, physischen Objekten, besitzen virtuelle Objekte standardmäßig nicht die Fähigkeit miteinander zu kollidieren. Ein virtuelles Objekt im virtuellen Raum ist ohne weitere Einflüsse nur ein dreidimensionales statisches Bild. Fähigkeiten wie Bewegung, Erkennen von Kollisionen, oder auch entsprechende Reaktionen auf Kollisionen müssen den virtuellen Objekten erst über Skripte zugewiesen werden.

3.7.1 Konzept

Erkennen von Kollisionen kann in MRK-Szenarien in vielerlei Hinsicht verwendet werden. Durch das Erkennen von Kollisionen können in der Simulation eines Arbeitsschrittes, mögliche Fehlerquellen aufgrund von Kollisionen von Robotern mit anderen Robotern oder mit Umgebungsobjekten, frühzeitig erkannt werden. Es können auch Zonen festgelegt werden. Jede Zone registriert über Kollisionen, wer oder was sich in ihr aufhält, wodurch sich der Zustand des Szenarios oder der Umgebung verändern kann. Zum Beispiel können alle Roboter ausgeschaltet werden, wenn sich ein Mensch in eine gefährdete Zone begibt.

3.7.2 Implementierung

Unity3D ist als Entwicklungsumgebung für dreidimensionale Spiele konzipiert, daher sind Möglichkeiten zur Erkennung von Kollisionen schon vorimplementiert [47]. Unity3D bietet dreidimensionale Kollisionsobjekte in den Formen Box, Kugel, Kapsel und *Mesh* (Maschennetz oder auch Polygonnetz), welche in Abbildung 23 zu sehen sind.

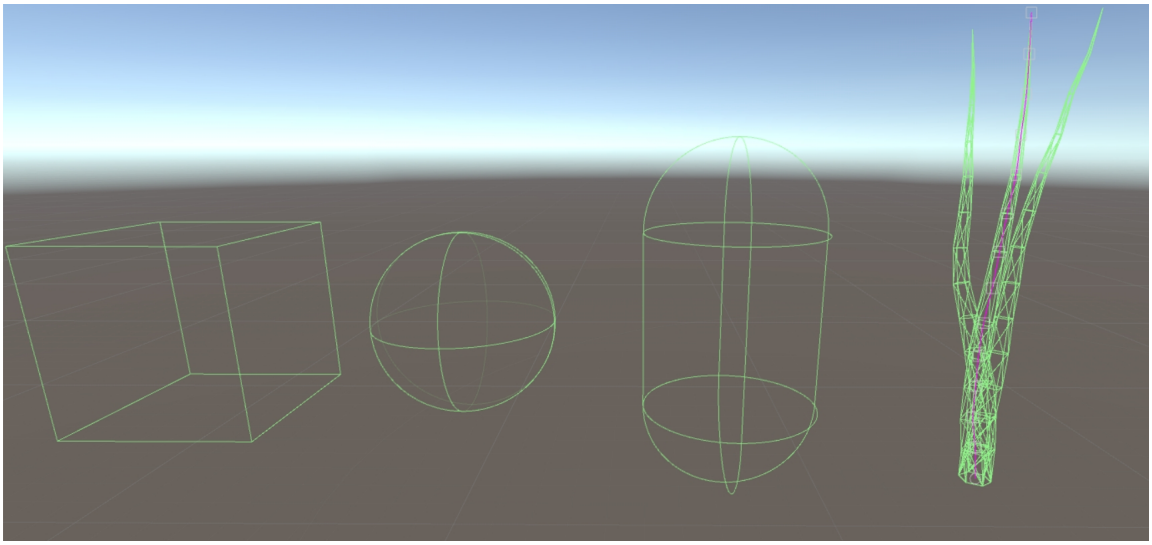


Abbildung 23: *Standardmäßig vorhandene Kollisionsobjekte in Unity3D. Von links nach rechts Box, Kugel, Kapsel und rechts ein Mesh-Kollisionsobjekt, hier angepasst an das Mesh von einer Pflanze*

Während die Form von Box-, Kugel- und Kapsel- Kollisionsobjekten eindeutig sind, passt sich die Form des *Mesh*-Kollisionsobjektes an das darunter liegende Polygonnetz an. Das Anpassen des *Mesh*-Kollisionsobjektes führt zu höherer Genauigkeit, ist jedoch nicht beweglich und auch aufwändiger in seiner Berechnung, weshalb bevorzugt mit Box oder Kugel-Kollisionsobjekten gearbeitet werden sollte. Unity3D Kollisionsobjekte können standardmäßig registrieren, wenn sie sich mit anderen Unity3D Kollisionsobjekten überschneiden, was einer Kollision entspricht. Kollisionsobjekte können mit virtuellen Objekten verknüpft werden, um diesen virtuellen Objekten die Kollisionseigenschaften der Kollisionsobjekte zur Verfügung zu stellen, wie in Abbildung 24 zu sehen ist. Wenn Physikeffekte aktiviert sind und angewendet werden, kann

[47] `https://docs.unity3d.com/ScriptReference/Collision.html`

eine Kollision auch dazu benutzt werden, dass sich Objekte voneinander abstoßen, wie in der realen Welt. Die Unity3D Kollisionsobjekte werden nicht nur für Kollisionen mit anderen Kollisionsobjekten verwendet, sondern auch für Raycasts 3.8. Raycasts ignorieren Objekte ohne Kollisionsobjekte und treffen nur auf Objekte mit aktivem Kollisionsobjekt. In einem MRK-Szenario können jedoch auch Objekte vorkommen, welche zwar Kollisionen erkennen können sollen, aber nicht für Raycasts zu erkennen sein sollen. Ein Beispiel hierfür ist eine Sicherheitszone, welche mehrere Roboter und Geräte umfasst. Wenn ein Mensch die Sicherheitszone betritt, sollen die Roboter zur Sicherheit des Menschen ausgeschaltet werden.

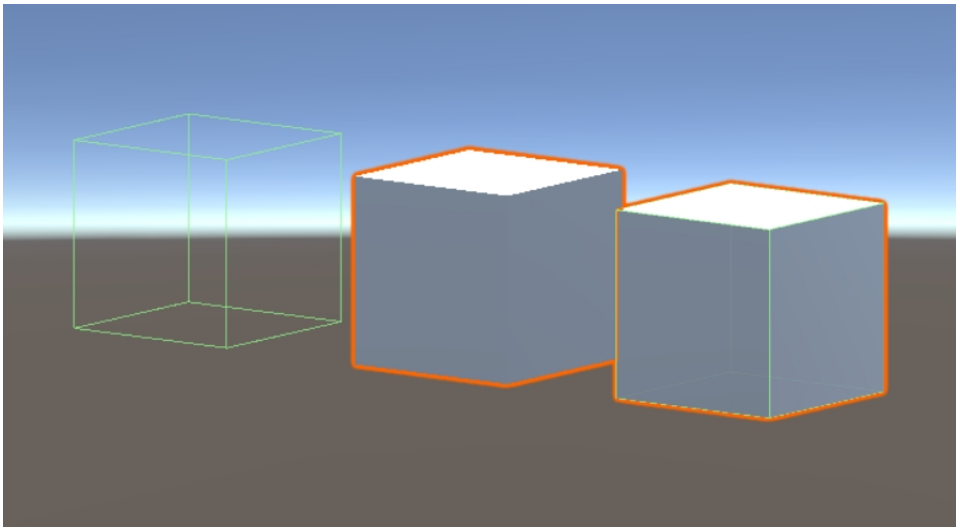


Abbildung 24: *Kollisionsobjekt (links), virtuelles Objekt ohne Kollisionsobjekt (mitte) und virtuelles Objekt verknüpft mit Kollisionsobjekt (rechts, erkennbar an den grünen Kanten) im Unity3D-Editor*

In diesem Aufbau ist es gewollt, dass der Raycast die Roboter in der Sicherheitszone trifft, ohne vorher von dem Kollisionsobjekt der Sicherheitszone abgefangen zu werden. Hierzu bietet Unity3D für Kollisionsobjekte verschiedene *Layer* (Ebenen) an, durch welche das Kollisionsobjekt der Sicherheitszone auf die *IgnoreRaycast* (Ignoriere Raycasts) Ebene gesetzt werden kann. Dadurch kann das Kollisionsobjekt der Sicherheitszone noch Kollisionen mit anderen Kollisionsobjekten registrieren, während Raycasts gleichzeitig das Kollisionsobjekt der Sicherheitszone ignorieren und wie gewünscht auf die Roboter in der Sicherheitszone treffen. Nur virtuellen Child-Objekten mit dem Stichwort *Collider* im Namen, werden die oben beschriebenen Kollisionsboxen automatisch hinzugefügt. Um eine fehlerfreie Ausführung des TEDURU-Servers zu gewährleisten, sollten dementsprechend auch keine weiteren virtuellen Child-Objekte beim Import in den TEDURU-Server über Kollisionsboxen verfügen.

3.8 Raycasting

Als Raycasting werden meistens Techniken zur schnellen Darstellung einer dreidimensionalen Szene bezeichnet, wobei heutzutage die genaue Definition des Begriffs kontextabhängig variiert [48]. Im Kontext von Unity3D wird damit eine Technik beschrieben, in der ein Strahl mit einer maximalen Länge von einem Ursprung ausgehend in eine Richtung geworfen wird, bis er mit einem Kollisionsobjekt 3.7 kollidiert [49].

3.8.1 Konzept

Eine weitere Anforderung an den TEDURU-Server ist die Unterstützung von Zeigegesten. Weiterhin soll es möglich sein zu erkennen, wo eine Person hinschaut. Durch das Wissen, wer oder was auf wen oder was zeigt, oder schaut können MRK-Szenarien, zum Beispiel durch den Gebrauch von multimodalen Dialogplattformen wie SiAM-dp [39] oder Anvil [29] erweitert werden.

3.8.2 Implementierung

Unity3D bietet ein Raycasting System für die oben beschriebenen Aufgaben. Hierzu wird vom TEDURU-Server einem virtuellen Child-Objekt mit dem *Raycaster* Stichwort im Namen ein Raycasterobjekt hinzugefügt. Dieses strahlt in die Y-Richtung des Objektes einen Raycast Strahl. Ein Raycast in Unity3D ist ein Strahl, welcher mit einer maximalen Länge, von einem Ursprung ausgehend in eine Richtung geworfen wird, bis er auf ein Kollisionsobjekt trifft. In Abbildung 25 wird dies anhand einer Zeigerichtung eines Roboters, als auch der Blickrichtung einer mobilen Roboterplattform demonstriert.

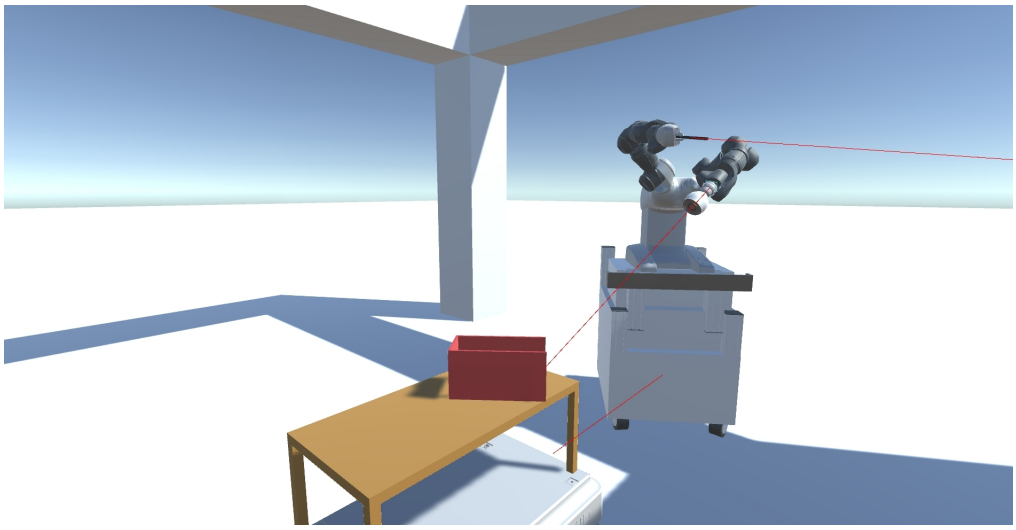


Abbildung 25: *Raycasts ausgehend von MiR100 mit einem auf-montierten Tisch in Fahrtrichtung (links) und ABB Yumi in Zeigerichtung der Arme (rechts). Raycasts werden zu Präsentationszwecken als rote Strahlen visualisiert.*

[48] `https://de.wikipedia.org/wiki/Raycasting`
[49] `https://docs.unity3d.com/ScriptReference/Physics.Raycast.html`

Ausnahme hierzu sind Kollisionsobjekte, welche sich auf der *IgnoreRaycast* Ebene befinden. Ein Raycast der auf ein Objekt trifft gibt das getroffene Objekt, sowie die Entfernung zum Objekt zurück. Wenn ein Raycast seine maximale Länge erreicht, ohne auf einen anderen Körper zu treffen, gibt er ein leeres Objekt zurück

3.9 Ankerpunkte

Ein Regal hat verschiedene Bretter und Fächer. Mit Ankerpunkten ist es im virtuellen Modell des Regals möglich die verschiedenen Bretter und Fächer zu annotieren. Virtuelle Objekte, die in das virtuelle Modell des Regals gelegt werden sollen, müssen nun nicht mehr manuell durch Berechnung der dreidimensionalen Position des gewünschten Fachs im Regal positioniert werden. Stattdessen kann das virtuelle Objekt, durch einen einfachen Befehl an die Position des Ankerpunktes gesetzt werden, der der Position des gewünschten Fachs im Regal entspricht.

3.9.1 Konzept

Virtuelle Objekte sollen vordefinierte Positionen besitzen können, welche direkt angesteuert werden können. Diese Positionen sollen Ankerpunkte heißen, da Objekte über sie an anderen Objekten verankert werden können.

3.9.2 Implementierung

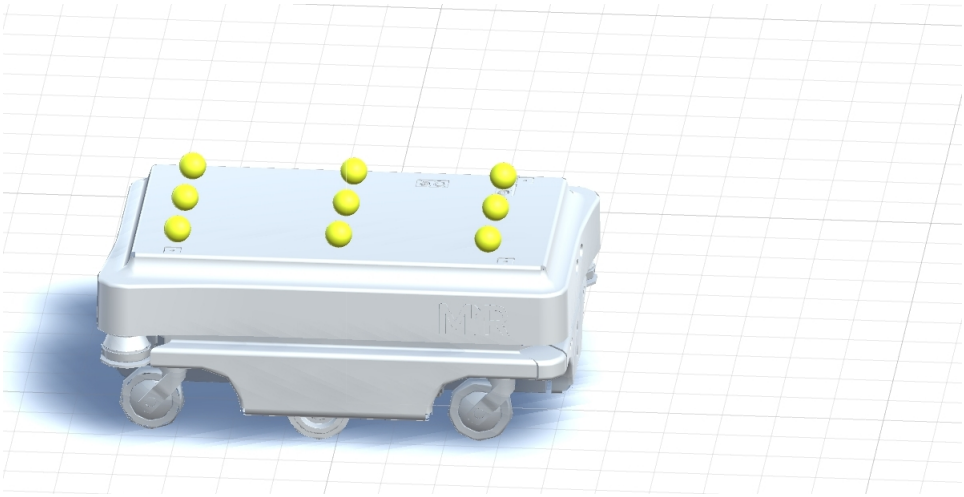


Abbildung 26: *MiR100 mit Ankerpunkten (gelbe Kugeln)*

Ankerpunkte sind genau wie Gelenke, Kollisionsobjekte und Raycasts, Funktionalitäten, die von dem TEDURU-Server annotierten virtuellen Objekten beim Laden automatisch zugewiesen werden. Um einem virtuellen Objekt die Funktionalität eines Ankerpunkts zuzuordnen, muss sein Name das Stichwort *Anchor* enthalten. In Abbildung 26 ist ein MiR100 mit neun Ankerpunkten (Ankerpunkte visualisiert durch gelbe Kugeln) zu sehen. Implementiert sind die Ankerpunkte lediglich dadurch, dass sie Komponenten eines Objektes sind, welche angesprochen werden können. Somit ist es über das *MoveTo* Event möglich, Objekte zu solch einem Ankerpunkt zu bewegen.

3.10 Hexagonkarten

Eine Hexagonkarte ist eine zweidimensionale Darstellung einer dreidimensionalen Umgebung. Hierbei wird die Umgebung in gleichgroße hexagonale Formen aufgeteilt. Jedes dieser hexagonalen Objekte weiß, ob es sich mit einem virtuellen Objekt überschneidet. Bei solch einer Überschneidung wird das entsprechende hexagonale Objekt als belegt markiert.

3.10.1 Konzept

Da in einem MRK-Szenario oft auch mobile Roboter eingesetzt werden, müssen diese ebenfalls in der Lage sein sich in der Umgebung zu bewegen. Wenn diese mobilen Roboter sich in der Umgebung bewegen sollen, benötigen sie eine Karte der Umgebung, auf der sie ihren Weg planen können, wie in Abbildung 3 zu sehen ist. Da es eine Aufgabe des TEDURU-Servers ist die physische Umgebung virtuell darstellen zu können, soll er auch weiterführend in der Lage sein eine zweidimensionale Karte der Umgebung zu generieren, welche von externen Anwendungen genutzt werden kann. Zusätzlich soll die Karte während der Laufzeit des Servers generiert werden, damit sie immer möglichst aktuell ist und nach Möglichkeit auf hexagonalen Flächen basieren. Hexagonale Flächen wurden hierbei den standardmäßig verwendeten rechteckigen Flächen vorgezogen. Dies hat mehrere Gründe. Zum einen ist die Laufzeit von A* Wegfindungsalgorithmen auf beiden Arten von Flächen gleich [40], wodurch in dieser Hinsicht kein Nachteil entsteht. Zum anderen bieten hexagonale Flächen, eine bessere topologische Repräsentation des zugrunde liegenden räumlichen Problems, da jede hexagonale Fläche abstandsgleich zu ihren benachbarten Flächen ist und mit jeder benachbarten Fläche eine Seite teilt. Außerdem gibt es modifizierte Formen von A*, wie zum Beispiel IDA* oder auch D*, die auf hexagonalen Flächen weit effektiver funktionieren als auf gewöhnlichen rechteckigen Flächen [55, 4].

3.10.2 Implementierung

Es wurden mehrere Lösungsansätze für das Erstellen einer Hexagonkarte in Betracht gezogen. So wurde überlegt, mit einer zusätzlichen virtuellen Kamera im TEDURU-Server, ein Bild des gewünschten Bereiches zu erstellen und dieses dann über ein Bildbearbeitungsprogramm in eine Hexagonkarte umzuwandeln. Ein weiterer Ansatz war es mehrere hexagonale Objekte, entsprechend der Maße der gewünschten Karte, zu Erstellen und über Kollisionen auf den hexagonalen Objekten, erfasst durch die Verwendung von Unity3D Kollisionsobjekten, zu bestimmen, welche Hexagone auf der Karte belegt sind und welche nicht. Implementiert ist der zweite Ansatz. Wie im Kapitel über Kollisionserkennung 3.7 beschrieben, verfügt Unity3D über einige Standardformen für Kollisionsobjekte. Diese umfassen für dreidimensionale Körper Box, Kugel, Kapsel und *Mesh* (Maschennetz), wie in Abbildung 23 zu sehen. Da in Unity3D jedoch keine hexagonal geformten Kollisionsobjekte standardmäßig vorhanden sind, muss die Kollision, mit den hexagonalen virtuellen Objekten, auf anderem Wege implementiert werden.

Der gewünschten hexagonalen Form am ähnlichsten sind die Unity3D Kapsel-Kollisionsobjekte. Hierbei muss jedoch mit kleineren Ungenauigkeiten bezüglich der rundlichen Form der Kapseln, gegenüber der gewünschten hexagonalen kantigen Form, gerechnet werden. Eine genaue Nachempfindung der hexagonalen Form ist durch Maschennetz-Kollisionsobjekte möglich, welche an das Maschennetz eines hexagonalen virtuellen Objektes angepasst ist. Wie schon vorher besprochen, ist die Berechnung einer Kollision über ein Maschennetz-Kollisionsobjekt jedoch aufwändiger, als die Berechnung einer Kollision auf einem einfachen Kollisionsobjekt.

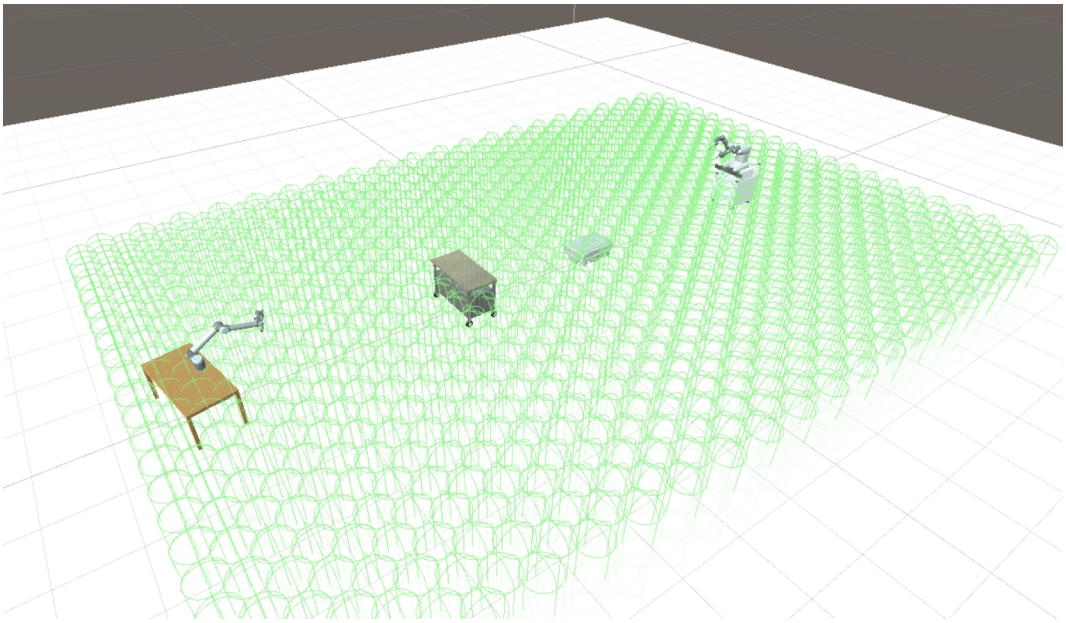


Abbildung 27: *Mehrere Kapselkollisionsobjekte, die zusammen im Virtuellen Raum eine Hexagonkarte aufspannen*

In Abwägung der beiden Möglichkeiten beruht die Implementierung schlussendlich auf Kapsel-Kollisionsobjekten, wie in Abbildung 27 zu sehen ist. Dies ergab sich daraus, dass fälschlicherweise belegte hexagonale Flächen auf einer Hexagonkarte, weniger ins Gewicht fallen, als erhöhter Rechenaufwand durch die Maschennetz-Kollisionsobjekte in hexagonaler Form, wenn der Radius der Hexagone entsprechend klein gewählt wird. Die Kollisionsobjekte der Hexagone sollen Raycasts nicht behindern, weshalb sie sich auf der *IgnoreRaycast* Ebene befinden.

3.11 Zusätzliche Informationen zu Objekten

Für viele MRK-Szenarien ist es nicht nur wichtig, dass die Positionen der Roboter über ein gemeinsames Weltbild synchronisiert sind, sondern auch, dass notwendige zusätzliche Hilfsinformationen zentral zur Verfügung gestellt werden.

3.11.1 Konzept

Der TEDURU-Server soll Möglichkeiten bieten, Objekte über kurze Merkmale in vordefinierte Gruppen einzuordnen. Zusätzlich sollte es über den TEDURU-Server möglich sein, die virtuellen Objekte mit zusätzlichen Informationen und Daten zu annotieren.

3.11.2 Implementierung

Da unterschiedliche Szenarien unterschiedliche Arten von Darstellungen für ihre Informationen benötigen, ist keine bestimmte Darstellung für Informationen implementiert worden. Stattdessen gibt es ein *AdditionalInfo* string Feld im TEDURU-Server, welches benutzt werden kann um Objekte beliebig zu annotieren. Zusätzlich wurde ein *Tag* Feld, welches mehrere

string-Objekte enthält umgesetzt. Das *Tag* Feld ist zur Gruppierung von Komponenten einer Umgebung angedacht. Zum Beispiel können Roboter den *Tag robot* und Menschen den Tag *human* erhalten. Wenn der TEDURU-Server eine Kollision von einer Komponente mit dem *Tag robot* und einer Komponente mit dem *Tag human* registriert, kann ein Alarm ausgelöst werden, welcher alle Roboter in dem Szenario per Notaus zum Stillstand bringt, um die Sicherheit der Menschen im Szenario zu gewährleisten.

3.12 Zurverfügungstellen von Informationen

Da der TEDURU-Server in der Art einer dreidimensionale Datenbank angelegt ist, muss es möglich sein, den aktuellen Zustand der virtuellen Umgebung des TEDURU-Servers nachzuschauen.

3.12.1 Konzept

Der TEDURU-Server benutzt das *network-toolkit TECS* mit *Publish Subscribe* (PS) als Kommunikationsmethode. Wenn ein Client aktuelle Informationen über den momentanen Zustand der virtuell simulierten Umgebung erhalten will, muss er eine Anfrage an den Server stellen und auf Antwort warten. Wenn in solch einer Umgebung mehrere Clients aktuelle Informationen über den momentanen Zustand der virtuell simulierten Umgebung erhalten müssen, kann der daraus erzeugte Kommunikationsaufwand an eingehenden Anfragen den Server überlasten. Insbesondere dann, wenn gleichzeitig auch neue aktuelle Positionen und Zustände von Agenten und Aktuatoren des Szenarios den Server erreichen sollen. Deshalb soll der TEDURU-Server eine einfache Möglichkeit bieten, Informationen über den aktuellen Zustand der Umgebung bereitzustellen.

3.12.2 Implementierung

Der TEDURU-Server enthält eine *Repeater* Klasse, welche in regelmäßigen Intervallen alle Informationen veröffentlicht, die nötig sind, um den aktuellen Zustand der Umgebung nachzuempfinden. Damit kleinere Anwendungen die nur bestimmte Informationen benötigen, um ihre Aufgabe zu erfüllen, nicht überlastet werden, wird der Zustand der virtuellen Umgebung des TEDURU-Servers in mehreren unabhängigen Datenpaketen übermittelt. Die Pakete, die der TEDURU-Server übermittelt, sind zum einen ein Inhaltspaket, ein Kollisionspaket, ein Raycast Paket und ein Paket für den Zustand der Hexagonalen Karten, sofern diese vorhanden sind. In dem Inhaltspaket enthalten sind alle Objekte der Szene, samt ihrer ID, Transformationsdaten, ihrer Tags und möglicher zusätzlicher Informationen, sowie das Entsprechende auch für alle Unterobjekte-Objekte, des Objektes. Zum anderen veröffentlicht der TEDURU-Server auch ein Kollisionspaket, so dass gemäß 3.7 alle nötigen Informationen zu Kollisionen zwischen Kollisionsobjekten enthalten sind. Als drittes veröffentlicht der TEDURU-Server ein Raycast Paket, das gemäß 3.8 alle Informationen über Raycasts in der virtuellen Umgebung enthält. Das vierte Paket verschickt entsprechend den Zustand aller aktiven Hexagonkarten.

Durch die Aufteilung der gesamten Informationen in drei kleinere Pakete können kleinere Anwendungen sich gezielt an der Informationsquelle anmelden, welche ihrem Aufgabengebiet entspricht. Zum Beispiel eine SmartWatch, deren einzige Aufgabe es ist, eine Warnung von sich zu geben, wenn ein Objekt mit dem *Tag human* mit einem virtuellen Objekt mit dem Tag *dangerzone* kollidiert. Diese SmartWatch meldet sich dann nur für TEDURU-Server Kol-

lisionspakete an, da alle weiteren Informationen nicht notwendig sind um die eigene Aufgabe zu erfüllen.

3.13 Benutzerinterface im Server

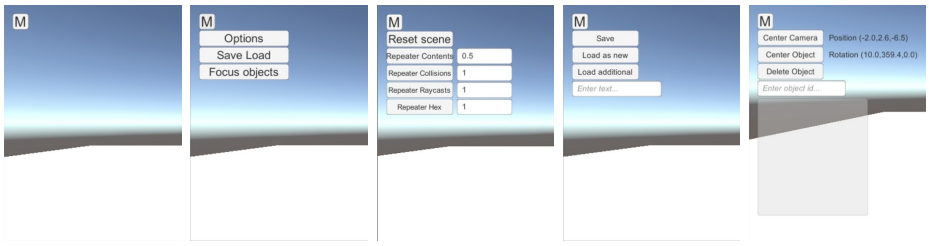
Ein Nutzer einer Anwendung, kann über ein Benutzerinterface mit der Anwednung interagieren.

3.13.1 Konzept

Der TEDURU-Server ist nicht nur ein Server zur virtuellen Darstellung und Simulation einer realen Umgebung, sondern auch eine Anwendung, mit der es möglich ist sich in dieser Umgebung umzusehen 3.2. Des Weiteren kann die virtuelle Szene, samt wichtiger Komponenten, auch gespeichert und geladen werden 3.6. Diese Funktionalitäten sollen möglichst einfach von einem Nutzer des Server abgerufen werden können.

3.13.2 Implementierung

Ziel ist es nicht, alle möglichen Interaktionsmögichkeiten mit dem TEDURU-Server, über das Benutzerinterface des TEDURU-Servers zugänglich zu machen, sondern ein Interface zu bieten, mit dem es einem Nutzer möglich ist, sich in der virtuellen Umgebung zurecht zu finden und kleinere Anpassungen direkt am Server zu treffen. Der TEDURU Server ermöglicht seinen Nutzern die virtuelle Kamera frei in der virtuellen Umgebung zu bewegen, wie schon im Absatz 3.2 beschrieben. Zusätzlich bietet der TEDURU-Server ein minimalistisches Interface, bestehend aus Untermenüs, wie in Abbildung 28 zu sehen ist.



(a) *TEDURU-Server Startmenü* (b) *TEDURU Menüauswahl* (c) *Menü Option: Options* (d) *Menü Option: Save Load* (e) *Menü Option: Focus Object*

Abbildung 28: *Startmenü samt Unteroptionen des TEDURU-Servers*

Unity3D verfügt über ein ausreichend großes Repertoire an Möglichkeiten um Menüs zu erstellen. Dies beruht auf der Herkunft Unity3Ds als Entwicklungsumgebung für Spiele. Diese Möglichkeiten wurden genutzt um ein Menü für den TEDURU-Server zu erstellen. Im Folgenden wird der Aufbau des Menüs beschrieben.

Damit das Menü nicht die Sicht auf das Geschehen blockiert, gibt es in der Ausgangsposition nur einen Knopf mit der Aufschrift **M** für Menü, in der oberen linken Ecke der Anwendung 28a. Durch einen Klick auf den Menüknopf erscheinen drei weitere Knöpfe, welche eine Menüauswahl darstellen 28b. Die Menüoptionen die ausgewählt werden können sind, *Options*, *Save Load* und *Focus Object*. Durch einen Klick auf die erste Option *Options* wird

ein Optionen Untermenü aufgerufen. Dort ist es sowohl möglich die virtuelle Umgebung im TEDURU-Server Neu zu starten, wodurch die Szene in einen leeren Ausgangszustand zurück gesetzt wird, als auch die Intervalle in denen die *Repeater* Klasse 3.12 ihre Informationen zur Verfügung stellt, zu verändern 28c. Mit der zweiten Option *Save Load* der Menüauswahl, wird das Speichern und Laden Untermenü aufgerufen 28d. Hier kann man den Zustand der aktuell abgebildeten virtuellen Umgebung abspeichern, sowie eine andere Szene als neues Projekt laden, oder auch eine vorher abgespeicherte Szene zusätzlich zu der aktuell laufenden Szene laden, um zum Beispiel zwei Projekte in einer Szene darzustellen. Über das Eingabefeld in diesem Untermenü wird der Name der zu speichernden Szene eingegeben, während das aktuelle Datum zur Speicherzeit mit an den Namen angehängt wird. Zum Laden einer Szene muss in das Eingabefeld der korrekte Name der abgespeicherten Datei eingegeben werden. Die dritte Option der Menüauswahl ist die *Focus Object* Option. In dem *Focus Object* Untermenü können Informationen über die Position und Rotation der Kamera abgerufen werden, sowie die Kamera auf ein Objekt in der Szene fokussieren, oder auch die Kamera an ihren Ausgangspunkt zurücksetzen 28e. Die Auswahl des zu fokussierenden Objektes findet über eine Eingabe der Objekt-ID in das Eingabefeld und einen anschließenden Klick auf den *Focus Object* Knopf statt. In dem transparenten, grauen Kasten werden anschließend einige Informationen über das fokussierte Objekt angezeigt. Um in dem Menü, von einem Untermenü in ein Obermenü zu navigieren muss lediglich wieder auf die M Schaltfläche oben links gedrückt werden.

4 Evaluation

Der TEDURU-Server ist ein Rahmenwerk für MRK-Szenarien und als solches ist es nur schwer möglich, den TEDURU-Server auf regulärem Weg über eine Benutzerstudie zu bewerten. Um dennoch eine Bewertung des TEDURU-Servers durchführen zu können, wurde eine Evaluation anhand eines MRK Szenarios durchgeführt. Hierzu ist das Szenario für zwei unterschiedliche Systeme entwickelt und implementiert worden. Zwei Kriterien wurden hierzu herangezogen, um aus diesem Experiment heraus zu bewerten, ob der TEDURU-Server in Bezug auf die Forschungsfragen dieser Bachelorarbeit ein Erfolg ist. Damit der TEDURU-Server als Antwort für die erste Forschungsfrage herangezogen werden kann, wurde eine Zeitmessung durchgeführt. Gemessen wurde die Arbeitszeit die notwendig war, um das Szenario, im Anschluss an eine erfolgreiche Implementierung mit dem ersten System, auch für das zweite System zu implementieren. In Anbetracht der zweiten Forschungsfrage sollte es möglich sein, das zweite System zu implementieren, ohne Änderungen an der Kommunikationsstruktur oder dem Aufbau des Szenarios vorzunehmen.

4.1 Einführung

Das Szenario, welches für diese Evaluation herangezogen wird, besagt, dass in einer Arbeitsumgebung mehrere stationäre Roboter sowie zwei mobile Roboterplattformen in Gebrauch sind. Der Bereich um jeden Roboter ist in zwei Gefährdungszonen eingeteilt. Die erste Zone ist eine Gefahrenzone und die zweite Zone ist eine kritische Zone. Die Gefahrenzone ist in einem Radius von etwa einem Meter um den möglichen Wirkungsbereich des Roboters, während die kritische Zone innerhalb des direkten Wirkungsbereiches des Roboters liegt. Wenn ein Arbeiter eine kritische Zone betritt, muss der entsprechende Roboter, dessen Gefahrenbereich übertreten wurde, deaktiviert werden, um die Sicherheit des Arbeiters gewährleisten zu können. Bei dem Betreten einer Gefahrenzone soll der Arbeiter darüber informiert werden, dass er sich in einem gefährdeten Bereich befindet. Auf der Abbildung 30b sind solche Zonen zu sehen. Die gelben Kästen spiegeln hierbei die Gefahrenzonen wider, während die roten Kästen in den Gefahrenzonen die kritische Zonen darstellen.

In der Umgebung des beschriebenen MRK-Szenarios soll ein Arbeiter, ausgestattet mit einem Ortungssystem, seiner Arbeit nachgehen können, ohne in die kritische Zone eines Roboters zu geraten. Hierzu muss er informiert werden, wenn er sich einer kritischen Zone eines Roboters nähert. Zusätzlich zu dem Szenario, sollen nach Möglichkeit der verwendeten Ortungssysteme noch zusätzliche Hilfsfunktionen implementiert werden, welche die Fähigkeiten des TEDURU-Servers repräsentieren und gleichzeitig das MRK-Szenario sinnvoll erweitern.

4.2 Motivation

Es muss ausgewertet werden, inwieweit der TEDURU-Server die Forschungsfragen dieser Arbeit beantwortet. Durch eine Umsetzung des beschriebenen MRK-Szenarios mit zwei unterschiedlichen Ortungssystemen für die Position des Arbeiters soll bewiesen werden, dass der TEDURU-Server über ein wiederverwendbares Protokoll für MRK-Szenarien verfügt. Durch einen geringen benötigten Zeitaufwand für das Umschreiben der Anwendung von einem Ortungssystem in ein anderes, wird bewiesen, dass der TEDURU-Server ein Dual Reality Rahmenwerk ist, mit dem es möglich ist, schnell Prototypen zu entwickeln und der TEDURU-Server gleichzeitig die Produktion beschleunigt. Wenn die Implementierung des Szenarios für

zwei unterschiedliche Ortungssysteme mit geringem Zeitaufwand möglich ist, wird somit nahegelegt, dass der TEDURU-Server als Antwort für die beiden Forschungszielen dieser Arbeit herangezogen werden kann. Die Ziele dieser Forschungsarbeit sind:

- Die Erschaffung eines Rahmenwerkes zur Vereinfachung und Beschleunigung der Erstellung von Prototypen und fertigen Produkten in MRK-Szenarien.
- Die Definition eines wiederverwendbaren Kommunikationsprotokolls für MRK-Szenarien.

4.3 Zielsetzung

Damit die Evaluation erfolgreich abgeschlossen ist, müssen folgende Bedingungen erfüllt werden. Das beschriebene Szenario muss funktionsfähig umgesetzt werden. Hierbei soll es in seiner ersten Implementierung mit einer Microsoft *Hololens* [50] als Ortungssystem und als Alarmsystem umgesetzt werden. Im Anschluss daran, muss in etwas weniger als einer Arbeitswoche das gleiche MRK-Szenario mit einem *Marvelmind* [51] als Ortungssystem und einer *Smartwatch* von *Androidwear* [52] als Alarmsystem umgesetzt werden. Das MRK-Szenario an sich ist zu simulieren, ohne physische Roboter in Betrieb zu nehmen, um den Aufwand der Evaluation auf das Wichtigste zu reduzieren und mögliche Versuchspersonen im Falle einer Fehlfunktion der Ortungssysteme, oder der Implementierung nicht zu gefährden.

4.4 Konzept

Da keine physischen Roboter in der Evaluation benutzt werden, ist die Platzierung der verschiedenen Gefahrenzonen für die Roboter an sich irrelevant. Jedoch sollte, aufgrund der für diese Arbeit zur Verfügung stehenden Industrieumgebung, diese auch entsprechend in die Evaluation eingebunden werden. Die Gefahrenzonen sollen deshalb, an die für die Dauer der Evaluation inaktiven Roboter, möglichst realitätsgenau angepasst werden. Die Ortung von Personen ist mit der Microsoft Hololens und mit einer Kombination des Marvelmind und der *LG Watch* umzusetzen. Da die Hololens im Gegensatz zum Marvelmind Ortungssystem auch ihre Rotation feststellen kann, ist dies zu nutzen, um auf der Hololens weitere Informationen, basierend auf der Blickrichtung des Trägers, anzuzeigen. Es soll für einen Nutzer der Anwendung, für beide Ortungssysteme auch leicht ersichtlich sein, ob er sich außerhalb jeglicher Gefahrenzone, in einer Gefahrenzone, oder in einer kritischen Zone befindet.

4.5 Implementierung

Um die virtuelle Simulation und die damit verbundene Platzierung der Zonen möglichst genau an den Begebenheiten der realen Umgebung zu halten, wurde auf einen vorher angefertigten dreidimensionalen Umgebungsscan und das daraus resultierende Umgebungsmodell, siehe Abbildung 29, zurückgegriffen.

Um die Zonen auf dem 3D Umgebungsmodell zu positionieren, wurde eine neue Anwendung mit nativer Unterstützung für das Format des 3D Scans, sowie mit Unterstützung für eine *network-toolkit TECS*-Anbindung, erstellt. Für diese Zonenanwendung wurde wiederum Unity3D benutzt, da Unity3D eine native Einbindung von diversen 3D Formaten bietet und

[50] `https://www.microsoft.com/de-de/hololens`
[51] `https://marvelmind.com/`
[52] `http://www.lg.com/de/wearables/lg-Watch-Urbane-2nd-Edition`

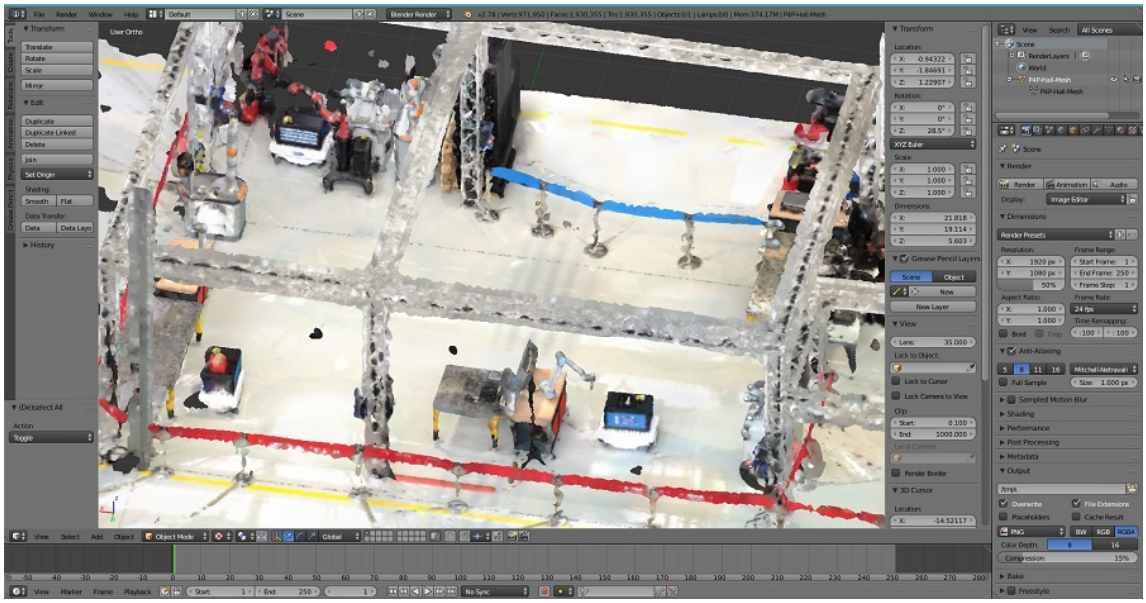
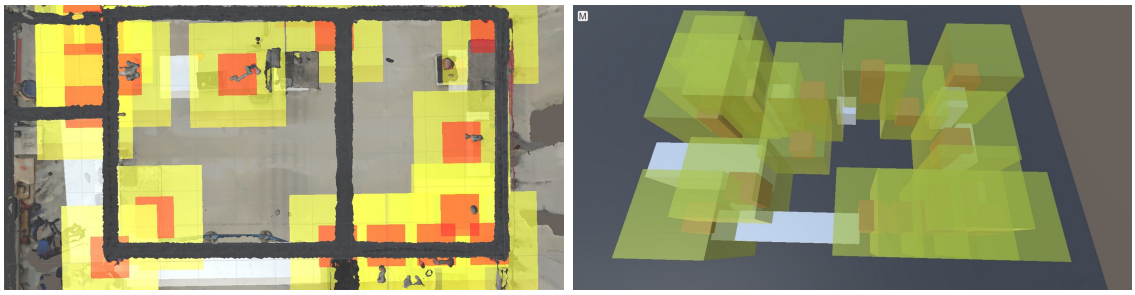


Abbildung 29: *3D Scan der für die Evaluation genutzten Umgebung. Visualisierung durch Blender.*

mit dem *network-toolkit TECS* kompatibel ist. Die Zonen an sich werden durch rechteckige Boxen dargestellt. Somit ist eine Einheit aus kritischer Zone, umgeben von einer Gefahrenzone, einfach durch zwei ineinander geschachtelte Boxen, darzustellen. Die verschiedenen Zonen wurden in der Editordarstellung von Unity3D, entsprechend des Umgebungsmodells, an die entsprechenden Positionen der Roboter und Sperrgebieten gesetzt, wie in Abbildung 30a zu sehen ist. Die Zonenanwendung übermittelt schlussendlich den Zustand der Zonen an den TEDURU-Server wie in Abbildung 30b zu sehen ist.



(a) *Zonenanwendung* (b) *TEDURU-Server mit Zonen*

Abbildung 30: *Zonenanwendung (links) und TEDURU-Server mit Zonen aus der Zonenanwendung (rechts). Gefahrenzonen sind gelb dargestellt und kritische Zonen sind rot dargestellt. Die weißen Zonen entsprechen Informationszonen.*

4.5.1 Hololens Implementierung

Die Hololens ist von Microsoft als tragbare *Augmented Reality* (AR) Brille konstruiert und verfügt daher standardmäßig über weitreichende AR Fähigkeiten. Zu den AR-Fähigkeiten der Hololens gehört zum Beispiel *Spatial mapping* [53]. *Spatial mapping* ermöglicht der Hololens eine Umgebung einzuscannen und ein Umgebungsmodell zu erstellen. Anhand der gesammelten Umgebungsdaten, kann die Hololens die eigene Position und Rotation in dieser Umgebung bestimmen. Des Weiteren kann die Hololens standardmäßig Hologramme [54] im AR-Sichtbereich der Brille darstellen, wie in den Abbildungen in 31 zu sehen ist.

(a) *KuKa Roboter und ein Roboterladebereich der durch mehrere Zonen abgesperrt ist*

(b) *UR10 mit seiner kritischen Zone, sowie angezeigtem Text*

Abbildung 31: *Sicht der Hololens auf die kritischen Zonen. Gefahrenzonen wurden nicht dargestellt, um sowohl den Nutzer, als auch die begrenzten Ressourcen der Hololens, nicht zu überbeanspruchen.*

Zusätzlich existiert eine gut ausgebaute und dokumentierte Unity3D-Hololens Schnittstelle [55]. Für den Hololens Teil wurde im Rahmen der Evaluation eine weitere Unity3D Anwendung implementiert, welche ihre Position und Rotation im Raum an den TEDURU-Server sendet, und die Position sowie die Kollisionen und Raycast Pakete des TEDURU-Servers empfängt. Mit den Positionen der Gefahrenzonen und kritischen Zonen stellt die Hololens die Zonen in ihrer eigenen AR Darstellung dar. Durch die Kollisionsinformationen aus dem TEDURU-Server kann die Hololens ermitteln, ob sie mit einer Gefahrenzone oder einer kritischen Zone kollidiert und entsprechend eines von zwei unterschiedlichen Warngeräuschen, wiedergeben. Aufgrund der AR Fähigkeiten der Hololens, werden die Alarmgeräusche räumlich simuliert, sodass ein Alarmgeräusch nicht nur ertönt, sondern auch aus der Richtung der Zone ertönt, mit der der Träger der Hololens kollidiert ist. Das Hololens-Modell auf dem TEDURU-Server ist mit einem Raycaster ausgestattet, wodurch es möglich ist, über Raycasts zu erfahren auf was die Hololens schaut. Hierzu befinden sich die Gefahrenzonen und die kritischen Zonen auf der *ignore Raycasts* Ebene 3.7, während Informationszonen mit zusätzlichen Informationen 3.11 versehen sind und per Raycast erfasst werden können. Somit kann auf der Hololens über ein Textfeld Informationen über das momentan angeschaute Objekt wiedergegeben werden, falls durch eine Informationszone Informationen hinterlegt sind.

[53] `https://developer.microsoft.com/en-us/windows/mixed-reality/spatial_mapping`
[54] `https://developer.microsoft.com/en-us/windows/mixed-reality/hologram`
[55] `https://unity3d.com/partners/microsoft/hololens`

4.5.2 Marvelmind + Smartwatch Implementierung

Abbildung 32: *Marvelmind Empfänger (links) Smartwatch (rechts). Die Smartwatch verfügt über drei Alarmstufen, grün (im Bild) kein Alarm, gelb in einer Gefahrenzone und rot in einer kritischen Zone.*

Marvelmind ist ein schall-basiertes Ortungssystem mit einer Genauigkeit von zwei Zentimetern, während die *Smartwacht* eine elektronische Armbanduhr mit zusätzlichen Sensoren, sowie Computerfunktionalitäten ist [56]. Zusammen bieten beide Systeme nicht die gleichen Möglichkeiten wie eine Hololens, trotzdem lässt sich das MRK-Szenario auch mit den beiden Systemen, anstelle einer Hololens, umsetzen. Die Marvelmind-Komponente übernimmt hierbei die Ortung des Arbeiters, während die *Smartwatch* den Arbeiter alarmiert, wenn er mit einer Zone in Berührung kommt, wie in Abbildung 32 zu erkennen ist. Da das Marvelmind-System nur die eigene Position in einer Umgebung, nicht aber seine Rotation bestimmt, ist die Implementierung von Raycasts und das daraus folgende Wiedergeben von zusätzlichen Informationen nicht möglich. Außerdem ist eine Implementierung von 3D Geräuschen über die *Smartwatch* nicht möglich, da die Position und Rotation der *Smartwatch*, innerhalb der Testumgebung, nicht erfasst werden kann. Somit sendet die Marvelmind Komponente nur die eigene Position an den TEDURU-Server, während die *Smartwatch* ausschließlich die Kollisionspakete des TEDURU-Servers über das *network-toolkit TECS* empfängt und auswertet. Am TEDURU-Server, oder an der Zonenanwendung, welche die Positionen der Zonen an den Server veröffentlicht, mussten keine Änderungen vorgenommen werden, damit das MRK-Szenario mit der Kombination aus Marvelmind und *Smartwatch* umzusetzen.

[56] `https://de.wikipedia.org/wiki/Smartwatch`

4.6 Ergebnis

Sowohl die Implementierung mit der Hololens, als auch die Implementierung mit der Kombination aus Marvelmind und *Smartwatch*, erfüllen die Anforderungen des MRK-Szenarios. Beide Implementierungen ermöglichen die Ortung eines Arbeiters und sie ermöglichen es auch den Arbeiter zu informieren, wenn er sich der kritischen Zone eines Roboters nähert. Die Implementierung des Szenarios mit dem Marvelmind und der *Smartwatch* wurde in weniger als einer Arbeitswoche abgeschlossen, was den Anforderungen der ersten Forschungsfrage entspricht. Zusätzlich waren keine Veränderungen der Kommunikationsstruktur oder des Aufbaus des Szenarios notwendig, was den Anforderungen der zweiten Forschungsfrage entspricht. Somit wird der TEDURU-Server, im Rahmen dieser Evaluation, beiden Forschungsfragen dieser Arbeit gerecht.

5 Konklusion

Wie aus der Evaluation ersichtlich wird, kann der TEDURU-Server als Hilfe herangezogen werden, um die zwei Forschungsziele dieser Bachelorarbeit zu erfüllen. Die beiden Ziele dieser Forschungsarbeit sind:

- *Die Erschaffung eines Rahmenwerkes zur Vereinfachung und Beschleunigung der Erstellung von Prototypen und fertigen Produkten in MRK-Szenarien*
- *Die Definition eines wiederverwendbaren Kommunikationsprotokolls für MRK-Szenarien.*

Um die erste Forschungsfrage zu beantworten, sollte es für ein beliebiges MRK-Szenario möglich sein, dass die Umsetzung des Szenarios mit dem TEDURU-Server schneller möglich ist, als eine Umsetzung ohne den TEDURU-Server. Der benötigte Zeitaufwand für solch eine Implementierung ist jedoch schwer durch eine Studie zu erfassen, da der benötigte Zeitaufwand stark von den Kenntnissen und Erfahrungen des implementierenden Programmierers abhängt. Deshalb wurde ein MRK-Szenario mit dem TEDURU-Server unter Verwendung von zwei unterschiedlichen Systemen implementiert und als Evaluationsmaßstab die Zeit gemessen, die benötigt wurde, um das Szenario für das zweiten System zu implementieren. Da die Implementierung des MRK-Szenarios mit dem zweiten System in weniger als einer Arbeitswoche möglich ist, kann der TEDURU-Server herangezogen werden, um die erste Forschungsfrage dieser Arbeit zu beantworten.

Um die zweite Forschungsfrage zu beantworten, muss gegeben sein, dass ein zentrales System, aus einem gegebenen Szenario entfernt und durch ein anderes unabhängiges System ersetzt werden kann, ohne dass Änderungen an der Kommunikationsstruktur oder dem Szenario vorgenommen werden müssen. Da in der Evaluation für den Umstieg von Hololens als Ortungs- und Alarmsystem, auf eine Kombination von Marvelmind als Ortungssystem und *Smartwatch* als Alarmsystem, keinerlei Änderungen an der Kommunikationsstruktur oder an dem Aufbau des Szenarios erforderlich waren, kann auch hier der TEDURU-Server herangezogen werden, um die zweite Forschungsfrage dieser Bachelorarbeit zu beantworten.

Durch die Evaluation ist somit gezeigt, dass der TEDURU-Server durchaus herangezogen werden kann, um die beiden Forschungsfragen dieser Arbeit zu beantworten. Somit kann der TEDURU-Server helfen, um Prototypen und Produktion zu vereinfachen und zu beschleunigen. Zusätzlich bietet der TEDURU-Server ein wiederverwendbares Protokoll für MRK-Szenarien.

6 Weiterführende Arbeiten

Der TEDURU-Server kann als Hilfe verwendet werden, um ein MRK-Szenario zu implementieren, oder Prototypen für solch ein Szenario zu erstellen. Er ist jedoch nicht dazu konzipiert worden, um als alleinstehende Anwendung ein solches MRK-Szenario lauffähig zu machen. Die Funktionalität des TEDURU-Servers entspricht deshalb zu Teilen mehr der einer dreidimensionalen Datenbank zum Abrufen von Information. Im Fall des TEDURU-Servers sind die Informationen, die abgerufen werden, Bestandteil einer virtuellen Umgebung, die den gesamten Zustand aller beteiligten Sensoren, Agenten und Aktuatoren einer realen Umgebung in einem CPE widerspiegelt. Der TEDURU-Server kann somit auch für Simulationen entsprechender Umgebungen benutzt werden. Der TEDURU-Server bietet zwar Speicherfunktionalität im Sinne einer Zustandsaufnahme, jedoch ist diese nicht für das Abspeichern und Wiedergeben von Simulationen, oder das Aufzeichnen von Interaktionen optimiert.

Das Benutzerinterface des TEDURU-Servers ist minimalistisch gehalten, so dass es nicht störend auffällt, oder die Performance des Servers beeinflusst. Jedoch gibt es Möglichkeiten das Interface zu erweitern und verbessern, so dass es sein minimalistisches Konzept beibehält, trotzdem aber in der Lage ist, mehr Informationen über die aktuelle Szene und deren Inhalt preiszugeben. So könnte ein neues Untermenü mit einer Auflistung von allen in der Szene vorhandenen virtuellen Objekten und deren IDs erstellt werden. Zusätzlich könnte noch die Möglichkeit eingebaut werden, den virtuellen Objekten zusätzliche Informationen, oder Annotationen direkt über das Benutzerinterface zuzuordnen oder auch die farbliche Darstellung der virtuellen Objekte im TEDURU-Server direkt verändern zu können, wodurch die Darstellung im Server bei abstrakten Modellen verdeutlicht werden würde.

Des Weiteren müssen virtuelle Objekte, die vom Server geladen werden sollen, als *AssetBundles* zur Verfügung gestellt werden. Einfacher wäre es in manchen Fällen jedoch, wenn der TEDURU-Server in der Lage ist, industriell gebräuchliche Objektbeschreibungen, wie zum Beispiel *urdf* [57] oder *sdf* [58] einzulesen und virtuelle Modelle aus ihnen zu erstellen, die ohne manuelle Nachbearbeitung vom TEDURU-Server genutzt werden können. Somit wäre es noch einfacher den TEDURU-Server zu nutzen, da die Modelle nicht mehr manuell annotiert werden müssen. Auch eine Konvertierung der virtuellen Modelle zu *AssetBundles* wäre damit überflüssig.

Der normale Unity3D Raycast, der vom TEDURU-Server verwendet wird, ist vergleichbar mit dem Strahl eines handelsüblichen Laserpointers. Er ist präzise und genau. Wenn es jedoch darum geht die Zeigegeste oder die Blickrichtung eines Menschen zu deuten, so wird ein einziger, dünner Strahl, ausgehend von dem Ursprung des Menschen, nicht unbedingt das gewünschte Objekt treffen oder nur eines, obwohl eine Gruppe von Objekten gemeint war. Um solch eine natürliche Zeigegeste mit dem TEDURU-Server zu simulieren, können unterschiedliche Ansätze verfolgt werden.

So könnte ein einzelner Raycast um mehrere Raycasts in trichterförmiger Anordnung erweitern werden, wobei sich der Umfang des Trichters an die Entfernung des zentralen Raycasts anpasst.

Bei einer Zeigegeste mit der Raycast-Trichter Herangehensweise werden alle Objekte, die von den Raycasts erfasst werden, mit ihrer globalen Position und Entfernung zum zentra-

[57] `http://wiki.ros.org/urdf`
[58] `http://www.sdformat.org/`

len Raycast zurückgegeben. Hierbei ist es dem Empfänger überlassen, die für ihn nötigen Informationen herauszufiltern.

Ein ähnlicher Ansatz für die Problematik ist, anstelle eines Raycasts ein zylinderförmiges Kollisionsobjekt, das alle Kollisionen, vom Ausgangspunkt bis zu einer Oberfläche auf die es trifft, zurückgibt, zu benutzen.

Beide Ansätze haben ihre Vor und Nachteile, würden jedoch eher einer natürlichen Zeige- oder Blickgeste entsprechen, als ein einziger Raycast. Eine weitere mögliche Hilfe für einen Benutzer des TEDURU-Servers wäre eine Möglichkeit aktive Raycasts und aktive Kollisionsboxen im TEDURU-Server visuell darzustellen.

Literatur

[1] Maribeth Back, David Arendash, James Vaughan, Anthony Dunnigan, Don Kimber, Thea Turner, and John Doherty. 2012. Design Evolution of a Mixed Reality Factory System. fxpal White Paper (2012).

[2] Maribeth Back, Don Kimber, Eleanor Rieffel, Anthony Dunnigan, Bee Liew, Sagar Gattepally, Jonathan Foote, Jun Shingu, and Jim Vaughan. 2010a. maribeth back the virtual chocolate facory;building a real world mixed-reality system for industrial collaboration and controlFXPAL-PR-10-568.pdf. In Multimedia and Expo (ICME), 2010 IEEE International Conference on. IEEE, 1160–1165.

[3] Maribeth Back, Don Kimber, Eleanor Rieffel, Anthony Dunnigan, Bee Liew, Sagar Gattepally, Jonathan Foote, Jun Shingu, and James Vaughan. 2010b. The virtual chocolate factory. In Proceedings of the international conference on Multimedia - MM '10. ACM, 1505. DOI:http://dx.doi.org/10.1145/1873951.1874262

[4] Yngvi Björnsson, Markus Enzenberger, Rob Holte, Jonathan Schaeffer, and Peter Yap. 2016. Comparison of Different Abstractions for Pathfinding on Maps. In The 19th International Joint Conference on Artificial Intelligence (IJCAI 03). 1511–1512.

[5] William Bluethmann, Robert Ambrose, Myron Diftler, Scott Askew, Eric Huber, Michael Goza, Fredrik Rehnmark, Chris Lovchik, and Darby Magruder. 2003. Robonaut: A robot designed to work with humans in space. Autonomous Robots 14, 2-3 (2003), 179–197. DOI:http://dx.doi.org/10.1023/A:1022231703061

[6] Maged N Kamel Boulos, Lee Hetherington, and Steve Wheeler. 2007. Second Life: An overview of the potential of 3-D virtual worlds in medical and health education. Health Information and Libraries Journal 24, 4 (2007), 233–245. DOI:http://dx.doi.org/10.1111/j.1471-1842.2007.00733.x

[7] B Brandherm and T. Schwartz. 2005. Geo referenced dynamic Bayesian networks for user positioning on mobile systems. In Location-and Context-Awareness. Springer, 223–234. DOI:http://dx.doi.org/10.1007/11426646_21

[8] Boris Brandherm, Sebastian Ullrich, and Helmut Prendinger. 2008. Simulation framework in second life with evaluation functionality for sensor-based systems. In CEUR Workshop Proceedings, Vol. 393.

[9] Manfred Broy. 2010. Cyber-Physical Systems. (2010). DOI:http://dx.doi.org/10.1007/978-3-642-14901-6

[10] Stefano Carpin, Mike Lewis, Jijun Wang, Stephen Balakirsky, and Chris Scrapper. 2007. USARSim: A robot simulator for research and education. In Proceedings - IEEE International Conference on Robotics and Automation. IEEE, 1400–1405. DOI:http://dx.doi.org/10.1109/ROBOT.2007.363180

[11] Balasubramaniyan Chandrasekaran and James M. Conrad. 2015. Human-robot collaboration: A survey. Conference Proceedings - IEEE SOUTHEASTCON 2015-June, June (2015), 47–66. DOI:http://dx.doi.org/10.1109/SECON.2015.7132964

[12] Beth Coleman. 2009. Using sensor inputs to affect virtual and real environments. IEEE Pervasive Computing 8, 3 (2009), 16–23. DOI:http://dx.doi.org/10.1109/MPRV.2009.60

[13] Diane J. Cook and Sajal K. Das. 2005. Smart Environments: Technology, Protocols and Applications. Vol. 43. John Wiley & Sons. 1–404 pages. DOI:http://dx.doi.org/10.1002/047168659X

[14] Jeff Craighead, Jennifer Burke, and Robin Murphy. 2008a. Using the Unity Game Engine to Develop SARGE : A Case Study. In Itsec, Vol. 4552. 366. http://www.robot.uji.es/research/events/iros08/contributions/craighead.pdf

[15] Jeff Craighead, Rodrigo Gutierrez, Jennifer Burke, and Robin Murphy. 2008b. Validating the search and rescue game environment as a robot simulator by performing a simulated anomaly detection task. In 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, Vol. 2006. 2289–2295. DOI:http://dx.doi.org/10.1109/IROS.2008.4650746

[16] Jeff Craighead, Robin Murphy, Jenny Burke, and Brian Goldiez. 2007. A survey of commercial & open source unmanned vehicle simulators. In Proceedings - IEEE International Conference on Robotics and Automation. IEEE, 852–857. DOI:http://dx.doi.org/10.1109/ROBOT.2007.363092

[17] František Duchon, Andrej Babinec, Martin Kajan, Peter Beno, Martin Florek, Tomáš Fico, and Ladislav Jurišica. 2014. Path planning with modified A star algorithm for a mobile robot. Procedia Engineering 96 (2014), 59–69. DOI:http://dx.doi.org/10.1016/j.proeng.2014.12.098

[18] ECMA Standard ECMA. 2001. 334: C# language specification. ECMA, December (2001).

[19] Daniel Ernst, Kimon Valavanis, Richard Garcia, and Jeff Craighead. 2007. Unmanned Vehicle Controller Design, Evaluation and Implementation: From MATLAB to Printed Circuit Board. Journal of Intelligent and Robotic Systems 49, 1 (2007), 85–108. DOI:http://dx.doi.org/10.1007/s10846-007-9130-4

[20] Jochen Frey, Robert Neßelrath, and Christoph Stahl. 2011. An Open Standardized Platform for Dual Reality Applications. In Proceedings of LAMDa, Vol. 11. 1–4.

[21] Eva Geisberger and Manfred Broy. 2012. Integrierte Forschungsagenda Cyber-Physical Systems - acatech STUDIE. München: TU, Institut für Informatik (2012).

[22] Brian P Gerkey, Richard T Vaughan, and Andrew Howard. 2003. The Player / Stage Project : Tools for Multi-Robot and Distributed Sensor Systems. In Proceedings of the International Conference on Advanced Robotics (ICAR 2003), Vol. 1. 317–323. DOI:http://dx.doi.org/10.1.1.10.491

[23] Lukas Hohl, Ricardo Tellez, Olivier Michel, and Auke Jan Ijspeert. 2006. Aibo and Webots: Simulation, wireless remote control and controller transfer. Robotics and Autonomous Systems 54, 6 (2006), 472–485. DOI:http://dx.doi.org/10.1016/j.robot.2006.02.006

[24] Aswin Indraprastha and Michihiko Shinozaki. 2009. The Investigation on Using Unity3D Game Engine in Urban Design Study. ITB Journal of Information and Communication Technology 3, 1 (2009), 1–18. DOI:http://dx.doi.org/10.5614/itbj.ict.2009.3.1.1

[25] ISO. 2008. IEC 24752: Information Technology—User Interfaces—Universal remote console—5 parts. Technical Report.

[26] Gene E. Jan, Ki Y. Chang, and I. Parberry. 2008. Optimal Path Planning for Mobile Robot Navigation. IEEE/ASME Transactions on Mechatronics 13, 4 (2008), 451–460. DOI:http://dx.doi.org/10.1109/TMECH.2008.2000822

[27] Gerrit Kahl. 2014. Dual Reality Framework : Basistechnologien zum Monitoring und Steuern von cyber-physischen Umgebungen. PhD-Thesis. Universität des Saarlandes. http://scidok.sulb.uni-saarland.de/volltexte/2014/5942/

[28] Pooya Karimian, Richard Vaughan, and Sarah Brown. 2006. Sounds good: Simulation and evaluation of audio communication for multi-robot exploration. In IEEE International Conference on Intelligent Robots and Systems. IEEE, 2711–2716. DOI:http://dx.doi.org/10.1109/IROS.2006.281995

[29] Michael Kipp. 2001. ANVIL A Generic Annotation Tool for Multimodal Dialogue. In Proceedings of the 7th European Conference on Speech Communication and Technology. 1367–1370.

[30] Antonio Krüger, Andreas Butz, CHristian Müller, Christoph Stahl, Rainer Wasinger, Karl-Ernst Steinberg, and Andreas Dirschl. 2004. The connected user interface: Realizing a personal situated navigation service. In IUI '04 Proceedings of the 9th international conference on Intelligent user interfaces. ACM, 161–168. DOI:http://dx.doi.org/10.1145/964442.964473

[31] Voemir Kunchev, Lakhmi Jain, Vladimir Ivancevic, and Anthony Finn. 2006. Knowledge-Based Intelligent Information and Engineering Systems. In International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, Vol. 4252. Springer, 537–544. DOI:http://dx.doi.org/10.1007/11893004

[32] Edward A. Lee. 2008. Cyber Physical Systems: Design Challenges. In 2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC). IEEE, 363–369. DOI:http://dx.doi.org/10.1109/ISORC.2008.25

[33] Joshua Lifton and Joseph A Paradiso. 2009. Dual Reality—Merging the Real and Virtual. In Springer LNICST, 1st International Conference on Facets of Virtual Environments (FAVE 09). Springer, 12–28.

[34] Joshua Harlan Lifton. 2007. Dual Reality : An Emerging Medium. Ph.D. Dissertation. Massachusetts Institute of Technology. http://resenv.media.mit.edu/pubs/theses/lifton

[35] Steve Mann. 1996. Smart clothing: the shift to wearable computing. *Commun. ACM* 39, 8 (1996), 23–24. DOI:http://dx.doi.org/10.1145/232014.232021

[36] O Michel. 2004. Cyberbotics Ltd - WebotsTM: Professional Mobile Robot Simulation. Inernational Journal of Advanced Robotic Systems 1, 1 (2004), 40–43.

[37] Paul Milgram, Paul Milgram, Herman Colquhoun, and Herman Colquhoun. 1999. A Taxonomy of Real and Virtual World Display Integration. Mixed Reality-Merging Real and Virtual Worlds 1 (1999), 5–30. DOI:http://dx.doi.org/10.1.1.32.6230

[38] R.R. Murphy, D. Riddle, and E. Rasmussen. 2004. Robot-assisted medical reachback: a survey of how medical personnel expect to interact with rescue robots. In Robot and Human Interactive Communication. IEEE, 301–306. DOI:http://dx.doi.org/10.1109/ROMAN.2004.1374777

[39] Robert Neßelrath. 2015. SiAM-dp : An open development platform for massively multimodal dialogue systems in cyber-physical environments. Ph.D. Dissertation. Saarbrücken, Universität des Saarlandes, Diss., 2016. http://scidok.sulb.uni-saarland.de/volltexte/2016/6385

[40] Mohd Fauzi Othman, Masoud Samadi, and Mehran Halimi Asl. 2013. Simulation of Dynamic Path Planning for Real-Time Vision-Base Robots. In Communications in Computer and Information Science, Vol. 376 CCIS. Springer, 1–10. DOI:http://dx.doi.org/10.1007/978-3-642-40409-2_1

[41] Joelle Pineau, Michael Montemerlo, Martha Pollack, Nicholas Roy, and Sebastian Thrun. 2003. Towards robotic assistants in nursing homes: Challenges and results. Robotics and Autonomous Systems 42, 3-4 (2003), 271–281. DOI:http://dx.doi.org/10.1016/S0921-8890(02)00381-0

[42] E.R. Post and M. Orth. 1997. Smart fabric, or wearable clothing. In Digest of Papers. First International Symposium on Wearable Computers. IEEE, 167–168. DOI:http://dx.doi.org/10.1109/ISWC.1997.629937

[43] Helmut Prendinger, Boris Brandherm, and Sebastian Ullrich. 2009. A Simulation Framework for Sensor-Based Systems in Second Life. Presence: Teleoperators & Virtual Environments 18, 6 (2009), pp. 468–477. http://web.ebscohost.com/ehost/detail?sid=eaf12261-7187-42e9-b3fd-fa58600fb80d@sessionmgr110

[44] Timothy Prestero. 2001. Development of a six-degree of freedom simulation model for the REMUS autonomous underwater vehicle. Ph.D. Dissertation. Massachusetts institute of technology. DOI:http://dx.doi.org/10.1109/OCEANS.2001.968766

[45] R Rusu, A Maldonado, and M Beetz. 2007. Extending player/stage/gazebo towards cognitive robots acting in ubiquitous sensorequipped environments. In ICRA Workshop for Networked Robot Systems, 2007, Rome, Italy. DOI:http://dx.doi.org/10.1.1.128.4033

[46] Christian Schonauer, Thomas Pintaric, Hannes Kaufmann, Stephanie Jansen-Kosterink, and Miriam Vollenbroek-Hutten. 2011. Chronic pain rehabilitation with a serious game using multimodal input. In 2011 International Conference on Virtual Rehabilitation, ICVR 2011. IEEE, 1–8. DOI:http://dx.doi.org/10.1109/ICVR.2011.5971855

[47] Mark Slee, Aditya Agarwal, and Marc Kwiatkowski. 2007. Thrift: Scalable cross-language services implementation. Facebook White Paper 5, 8 (2007), 8.

[48] Feijun Song, P. Edgar An, and Andres Folleco. 2003. Modeling and simulation of autonomous underwater vehicles: Design and implementation. IEEE Journal of Oceanic Engineering 28, 2 (2003), 283–296. DOI:http://dx.doi.org/10.1109/JOE.2003.811893

[49] Christoph Stahl. 2009. Spatial Modeling of Activity and User Assistance in Instrumented Environments. PhD-Thesis. Saarland University, Dept. of Computer Science, Saarbrücken. http://urn:nbn:de:bsz:291-scidok-34312http://scidok.sulb.uni-saarland.de/volltexte/2010/3431/https://www.dfki.de/web/forschung/publikationen/renameFileForDownload?filename=Dissertation-Final-Pflicht-Color.pdf&file_id=uploads_911

[50] Christoph Stahl, Jochen Frey, Jan Alexandersson, and Boris Brandherm. 2011. Synchronized realities. Journal of Ambient Intelligence and Smart Environments 3, 1 (2011), 13–25. DOI:http://dx.doi.org/10.3233/AIS-2011-0093

[51] Christoph Stahl and Jens Haupert. 2006. Taking location modelling to new levels: A map modelling toolkit for intelligent environments. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) 3987 LNCS (2006), 74–85. DOI:http://dx.doi.org/10.1007/11752967_6

[52] J Van Lawick-Goodall. 1971. Tool using in primates and other vertebrates. Advances in the Study of Behavior 3 (1971), 195–250.

[53] Steven Warburton. 2009. Second Life in higher education: Assessing the potential for and the barriers to deploying virtual worlds in learning and teaching. British Journal of Educational Technology 40, 3 (2009), 414–426. DOI:http://dx.doi.org/10.1111/j.1467-8535.2009.00952.x

[54] Mark Weiser. 1999. The computer for the 21 st century. ACM SIGMOBILE Mobile Computing and Communications Review 3, 3 (1999), 3–11. DOI:http://dx.doi.org/10.1145/329124.329126

[55] Peter Yap. 2002. Grid-based path-finding. In Conference of the Canadian Society for Computational Studies of Intelligence. Springer, 44–55.

[56] Andreas Zimmermann, Andreas Lorenz, and Reinhard Oppermann. 2007. An Operational Definition of Context. Modeling and Using Context 7 (2007), 558–571. DOI:http://dx.doi.org/10.1007/978-3-540-74255-5_42