Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

**Bachelorarbeit**

Christoph Hentschel

# Integration realer und virtueller Roboter in einer verteilten Mixed-Reality-Umgebung

*Fakultät Technik und Informatik*
*Department Informatik*

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

Christoph Hentschel

# Integration realer und virtueller Roboter in einer verteilten Mixed-Reality-Umgebung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Wolfgang Renz
Zweitgutachter: Prof. Dr. rer. nat. Michael Neitzke

Abgegeben am 10. November 2008

**Christoph Hentschel**

**Thema der Bachelorarbeit**
Integration realer und virtueller Roboter in einer verteilten Mixed-Reality-Umgebung
**Stichworte**
mixed reality, virtual reality, 3D, umgebung, agent, roboter, vehikel, kommunikation, message queue, simulation
**Kurzzusammenfassung**
Der Aufbau eines lose gekoppelten Systems zur Durchführung einer zeitlich versetzt oder in Echtzeit ablaufenden virtuellen Simulation wird beschrieben. In der virtuellen Welt können Agenten Vehikel steuern und externe Systeme Positionsdaten sich in der Realität bewegender Roboter anliefern. Die virtuelle Simulation bekommt die Befehle und die Positionsdaten als Nachrichten in eine Message-Queue geliefert, die die Persistenz und sequentielle Ordnung der Nachrichten garantiert. Die Simulation ist in der Lage bei der Auswertung der Nachrichten Ereignisse wie Kollision zu erfassen und an die Agenten und Roboter an einer von ihnen bestimmbaren Antwort-Message-Queue zurück zu liefern. Dadurch wird die Wahrnehmung der Roboter ihrer unmittelbaren Umgebung um die der virtuellen Umgebung erweitert. Die Vorgänge können auf dem Computerbildschirm aus einer in 3D positionierbaren Perspektive verfolgt werden.

**Christoph Hentschel**

**Title of the paper**
Integrating real and virtual robots in a distributed Mixed-Reality-Environment
**Keywords**
mixed-reality, virtual reality, 3D, environment, agent, robot, vehicle, communication, message queue, simulation
**Abstract**
The structure of a loose coupled system is presentend, which enables agents to steer virtual vehicles and external systems to provide current position data of moving robots to a virtual Simulation, both delayed or in real time. The virtual simulation gets instructions and position data supplied as messages in a message-queue, which guarantees persistence and sequential order of the messages. The Simulation evaluates the messages in the queue and is able to detect events such as collisions, which then can be supplied to the agents and robots at a reply message-queue of their choice, thus augmenting the perception of the robots of their next surroundings with that of the virtual environment. The action is presented on a computer screen from a 3D positionable perspective.

# Inhaltsverzeichnis

# Teil I

# Einleitung

# Kapitel 1

# Motivation

Die Erkundung unerschlossener Gebiete, einige davon ausserhalb unseres Planeten, erfolgt heute mit speziell ausgerüsteten Vehikeln. Wo möglich wird ein menschlicher Pilot an Bord des Vehikel die Steuerung übernehmen. Es gibt aber einige Gründe um auf die Transporttauglichkeit eines Menschen zu verzichten, zum Beispiel Energieeffizienz. Unbemannte Vehikel können einge Jahre passiv bleiben bis das zu erkundende Ziel erreicht wird, was wichtige Antriebsenergie für die letzendliche Aufgabe spart. Prominente unbemannte Vehikel sind das Jules Verne ATV[1] und der Mars Rover[2].

Unbemannte Vehikel werden in der Regel auf zwei Arten gesteuert: Per Fernsteuerung durch Menschen und/oder durch eine Steuerungssoftware. Während die Steuerungssoftware eines Vehikels in einem Laborpraktikum noch recht einfach sein kann, ist die autonome Steuerung eines Flugzeugs oder einer Sonde im Weltall äußerst Komplex. Völlig selbständig agierende Vehikel werden gemeinhin auch Roboter genannt und deren Steuerungssoftware eine künstliche *Intelligenz* zugesprochen.

Einen heute bevorzugten Ansatz bei der Entwicklung intelligenter Systeme stellen Agenten dar. Gemäß BDI Architektur [3] muss ein Agent erst mit Plänen versorgt werden, damit er diese zur Lösung ihm vorgegebener Aufgaben in unbekannten Situationen einsetzen kann. Von Kreativität kann jedoch nicht die Rede sein, dazu müsste ein Agent nicht nur Lernen sondern das Gelernte auch semantisch korrekt zur Lösung völlig neuer Aufgaben anwenden können.

Agenten eignen sich als autonome Steuereinheit(en) eines Vehikels erst dann, wenn sie alle Fähigkeiten eines Vehikels benutzen können, um übergeordnete Ziele zu erreichen. Zur Frage, wie Agenten die Fähigkeiten eines Vehikels nutzen sollen, sind bereits im Vorfeld zwei Designs vorstellbar:

1. Das Vehikel wird von einem vielseitigen Agent gesteuert.

[1] http://www.dlr.de/Portaldata/19/Resources/dokumente/atv_dlr_de_en.pdf

[2] http://marsrovers.jpl.nasa.gov/home/index.html

[3] BDI steht für Beliefs-Desires-Intentions

2. Ein Multiagent-System steuert das Vehikel. Jeweils ein Agent steuert nur einen bestimmten Teil des Systems. Die Agenten verstehen sich als Einheit, als *Leviathan*.

Die Fähigkeiten eines Vehikels zu kontrollieren lernen Agenten aber erst, wenn Programmierer ihnen vorkompilierte Pläne zur Verfügung stellen.

Durch die Gestaltung konkreter Aufgaben können solche Pläne entwickelt werden. Beziehen sich die Aufgaben dabei auf die Kontrolle oder Zusammenspiel mit Robotern, beschränkt sich die Auswahl der Aufgaben in der Regel auf

- die Anzahl der erworbenen oder produzierten Roboter und
- die Möglichkeiten der dafür vorgesehenen Räumlichkeiten

Hier können virtuelle Simulationen mehr Flexibilität bei der Gestaltung der Aufgaben bieten. Roboter müssen dazu aber erst in die virtuelle Simulation integriert werden. Dies lässt sich auf zwei Wege erreichen:

1. Der Roboter wird virtualisiert. Die Hardware wir per Software nachgebildet.
2. Der Roboter kommuniziert direkt mit der virtuellen Simulation.

Die Virtualisierung eines Roboters ist äußerst aufwändig. Ohne akkurater Nachbildung aller Bauteile, deren Eigenschaften und Bewegungsabläufe ist ein Simulationsmodell des Roboters nur von begrenzter Aussagefähgikeit.

Die zweite Alternative klinkt die Roboterhardware in die Schleife der virtuellen Simulation ein, was auch als *Hardware-in-the-Loop* bekannt ist. Einen Schritt weiter geht die *Erweiterung* der Wahrnemung des Roboters durch Ereignisse in der virtuellen Simulation.

# Kapitel 2

# Zielsetzung unf Problematik

## 2.1 Visualisierung von MAS:

Durch die in dieser Arbeit erzielten Erkenntnisse soll die Entwicklung visueller MAS Anwendungen ermöglicht werden, in denen die Umgebung der Agenten interaktiv geändert werden kann, um mit der Anpassungsfähigkeit von Agenten zu experimentieren und die Ergebnisse erfassbarer zu machen. Nehmen wir als Beispiel ein strategisches Computerspiel, in dem die Einheiten des künstlichen Gegners von einem MAS gesteuert werden. Auf bestimmte Aktionen des menschlichen Spielers lassen sich am Bildschirm die Reaktionen der einzelnen Agenten, die eine Einheit steuern, verfolgen. Für den Betrachter kann diese immersive Erfahrung nachvollziehbarer sein, als zum Beispiel die Entscheidungen der Agenten auf einer Konsole zu verfolgen.

## 2.2 Simulation:

Simulationen sind ein etabliertes Werkzeug bei der Entwicklung neuer Systeme.

> In der Industrie heißt die Devise Verkürzung der Entwicklungszeiten, und die Schlagworte dazu sind: Simultaneous Engineering und Virtual Prototyping (Preissler, 1995).

Durch Simulationen können Hardware und Steuerungssoftware kostengünstig erprobt werden. Risiken wie Hardwareverlust oder Gefährdung von Menschen können vermieden werden. Simulationen können real, virtuell oder gemischt ablaufen. Rein virtuelle Simulationen weichen allerdings oft stark von der Realität ab. Der Aufwand, der dafür getrieben wird, weist auf die erhebliche Differenz zwischen Simulation und Wirklichkeit hin. Dagegen sind die bei Augmentierung einer realen Simulation durch VR gewonnenen Erkenntnisse aufschlussreicher (Preissler, 1995).

## 2.3 Koppelung von Forschungsschwerpunkten:

Einige Schwerpunkte im Fachbereich Technik und Informatik der HAW sind:

1. *Ubiquitous computing.* Das Ubicomp-Lab widmet sich dem ubiquitären Computing, speziell Software-Architekturen für verteilte Systeme[4].

2. *Fahrerassistenz- und Autonome Systeme* In den FAUST-Projekten werden Technologien für Fahrerassistenz- und Autonome Systeme entwickelt[5].

3. *Labor für multimediale Systeme* MMLab: Entwicklung von VR-Anwendungen im Fred-Brooks-Labor. Weitere Themengebiete sind die Agenten-Orientierte Software-Entwicklung (AOSE), das Softwareengineering von Selbstorganisierenden Multiagentensystemen (SEsomas) sowie die Integration verteilter KI auf Basis von Agenten in VR und Spiele.

Ein Ziel dieser Arbeit ist es Wege zur losen Kopplung von Systemkomponenten, die in den einzelnen Forschungsschwerpunkten entwickelt werden, zu finden.

## 2.4 Problemstellung

Im Rahmen dieser Arbeit soll die Machbarkeit eines Systems untersucht werden, welches die lose Koppelung virtueller und realer Roboter in eine Virtual-Reality-Simulation ermöglicht. Den zeitlich und räumlich getrennten Teilnehmern der Simulation sollen Informationen zu Ereignissen bereit gestellt werden, die sich aus dem Zusammenspiel in der virtuellen Umgebung ergeben.

[4] http://www.informatik.haw-hamburg.de/uc.html
[5] http://www.informatik.haw-hamburg.de/faust.html

# Kapitel 3

# Inhaltlicher Aufbau

Grundlagen

: Einführung der verwendeten Begriffe und Zusammenhänge

Anwendungsfälle

: Wer kann das System zu welchen Zwecken nutzen

Anforderungen

: Ableitung von Anforderungen aus den Anwendungsfällen

Analyse

: Neben Literaturrecherche, Untersuchung der Anwendbarkeit bekannter Muster

: Begutachtung existierender Technologien

Architektur

: Architekturentwurf und vergleichbare Architekturen

Realisierung

: Performancemessungen

: Systemdokumentation

## 3.1 Zielpublikum

Zielpublikum sind die Entwickler von Virtual-Reality-, Multi-Agent und autonomen Systemen.

# Teil II

# Hauptteil

# Kapitel 4

# Grundlagen

Im Folgenden werden die dieser Arbeit zu Grunde liegenden Begriffe und Zusammenhänge erklärt.

## 4.1 Virtual Reality

*VR* versteht sich als Aufbau von Hard- und Software Komponenten, die dem Betrachter eine meist interaktive, nachvollziehbare Nachahmung der Realität präsentieren. Beim Zusammenschluss einer VR Anlage mit virtueller und/oder mechanischer Simulation ergeben sich Anwendungen wie

- Technologie gestützte Ausbildungssysteme an der Mensch/Maschine-Schnittstelle (Helmchen, 1995)
- 3D basierendes Virtual Prototyping von Mensch-Maschine Schnittstellen, ein neuer Ansatz zur Optimierung des Entwicklungsprozesses von Cockpitsystemen (Sabeur, 2003)
- Der neue BMW - Schwingungs- und Akustiksimulator, Aufbau und Anwendungen (Kirchknopf und Witta, 2003)

unter vielen anderen. In den meisten VR Anwendungen übernehmen *Renderingengines* softwareseitig die 3D Darstellung der Umgebung und der sich darin befindlichen Objekte. *Dynamics- oder Physicsengines* simulieren die physikalischen Einflüssen innerhalb der virtuellen Welt.

### 4.1.1 Visualisierung

Visualisieren bedeutet, abstrakte Daten oder Zusammenhänge in eine graphische beziehungsweise visuell erfassbare Form zu bringen. 3D-Visualisierung wird mit Hilfe von

*Rendering-APIs* umgesetzt. Die populärsten APIs sind OpenGL und DirectX. OpenGL besitzt eine offene Spezifikation und ist auf einer Vielzahl von Architekturen und Systemen implementiert. DirectX ist nur auf Betriebssysteme und Konsolen der Firma Microsoft implementiert. Einfacher zu nutzen sind darauf aufbauende *Szenengraphen*, die die objektorientierte Beschreibung des Zusammenhangs einer Szene in der virtuellen Welt anhand eines Graphenmodels erlauben (Davison, 2005).

#### 4.1.1.1 Funktion einer Renderingengine

Betrachten wir den Bildschirm als eine Leinwand oder *Canvas* auf die Punkte (in Form von *Pixel*) gezeichnet werden können. Ein-Punkt-Objekte werden auch Partikel genannt. Punkte lassen sich anhand ihrer Koordinaten durch einen *Ortsvektor* beschreiben. Objekte können aus mehreren Partikeln bestehen. Die Gerade zwischen zwei Partikel, so zu sagen die Kante eines Objekts, lässt sich als die Summe zweier Ortsvektoren ausdrücken. Setzt man die Anfangspunkte zweier Vektoren zusammen, wird eine Ebene gespannt. Beschränkt man die Ebene zusätzlich durch den *Summenvektor*, erhält man ein Dreieck, dessen Fläche als das *Kreuzprodukt* der Vektoren berechnet werden kann. Partikel und (Dreiecks-)Flächen sind gewissermaßen der kleinste gemeinsame Nenner für die Beschreibung von 3-D Objekten. Die Objekte lassen sich auf der Leinwand unterschiedlichen Perspektiven betrachtet werden. Dies wird auch *Projektion* genannt (Buss, 2003).

Jedes 3D Objekt besitzt neben Punkten und Flächen eine Position, den *Ortsvektor* des Stützpunktes. Wohin ein Objekt gerichtet ist, zeigt der *Richtungsvektor*, ausgehend vom Stützpunkt des Objekts. Durch die Operationen *Translation*, *Skalierung* und *Rotation*, allesamt als effiziente Matrizenmultiplikation implementierbar, lassen sich Position, Größe und Orientierung von 3D Objekten *transformieren*. Anstatt hier diese Operationen im Detail zu beschreiben ( unter anderem wie wir von Vektoren auf Matrizenmultiplikation kommen ) sei hier auf das WWW-basierte Lernprogramm Computergrafik Interaktiv - Grafiti[6] der Universität Oldenburg hingewiesen.

#### 4.1.1.2 Szenengraphen

sind gerichtete azyklische Graphen, deren Wurzel ein *virtuelles Universum* ist. Der Graph kann unterschiedliche Knotentypen enthalten, darunter einfache Verzweigungsknoten (BranchGroups) und transformierbare Knoten (TransformGroups). Ein Ast endet zum Beispiel in einer geometrischen Form oder einer Kameraperspektive (ebenfalls abstrahierte Objekte).

Bei der Entwicklung mit Szenengraphen beschreibt man den Zusammenhang einer Szene und bewirkt zu einem Zeitpunkt $t_0$ eine Transformation $T$ auf einen Knoten $K$ . Der Szenengraph übernimmt automatisch die Berechnung und Anwendung von $T$ auf $K$ und alle

[6] http://olli.informatik.uni-oldenburg.de/Grafiti3/index.html

Unterknoten $k_n$. Der Szenengraph sammelt alle nötigen Operation für die Renderingpipeline, optimiert sie und gibt sie dann an die unterliegende API (zum Beispiel OpenGL) weiter. Ein weiterer Vorteil (einiger) Szenengraphen ist die Austauschbarkeit der Renderingpipeline (Davison, 2005).

Es gibt kein allgemein gültiges Model eines Szenengraphen, da es in jeder Engine anders interpretiert und implementiert wird. Abbildung 4.1 aber veranschaulicht den prinzipiellen Aufbau anhand des Java3D Szenengraphen[7].

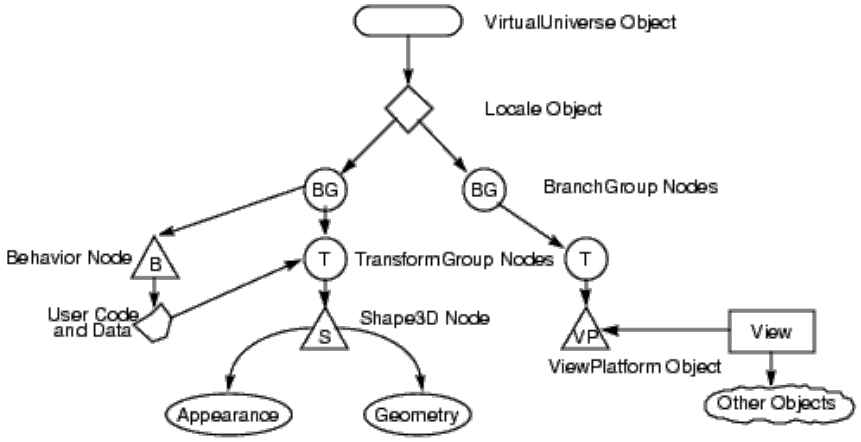


Abbildung 4.1: Java3D Szenengraphen

**Beispiel:** Ein transformierbarer Knoten $T$ enthält fünf weitere transformierbare Knoten $T_1, ..., T_5$. In $T_1$ ist das 3D Modell eines Fahrzeugs ohne Reifen enthalten. Die restlichen $T_{n-1}$ Knoten enthalten die geometrische Form eines Reifens. Durch Setzen unterschiedlicher Translationen ausgehend vom Ursprung $(0,0,0)$ für jedes $T_n$ können die Formen so positioniert werden, dass wir ein Fahrzeug mit vier Reifen an den richtigen Stellen erhalten. Wir erstellen nun die neue Transformation:

```
x=z=0, y=1;
t3d = new t3d();
t3d.transl( x, z, y );
t = T.getTransf();
```

Durch Multiplizierung von t.mul( t3d ); verschieben wir t um die Translation von t3d. Setzen wird das wieder in T ein, T.setTransf( t ), bewegt sich das Fahrzeug samt Reifen um $1$ in

[7] http://download.java.net/media/java3d/javadoc/1.5.2/javax/media/j3d/doc-files/intro.html#Structuring

Richtung $Y$. Allerdings drehen sich die Reifen dabei nicht. Hierfür müssten wir Rotationen auf die Knoten $T2, ..., T5$ kontinuierlich anwenden[8].

#### 4.1.1.3 Renderloop

Die Bilder einer Szene werde zyklisch in der so genannten *Renderloop* berechnet und auf den Bildschirm gebracht werden. Die Zahl der Schleifendurchgänge pro Sekunde ist die entscheidende Größe bei der Darstellung. Gemessen wird sie in *Frames-Per-Second* (Bilder pro Sekunde), kurz *FPS*. Ist die Anzahl der FPS zu hoch, kann das menschliche Auge der Szene nicht folgen. Ist die Anzahl der FPS zu niedrig, „stottert" das Bild und wirkt unnatürlich. Ein etabliertes Mindestmaß sind 60 FPS, was sich auf die Bildwiederholrate heute gängiger LCD Displays zurückführen lässt. Die Zeit zwischen den Bildern kann eine Anwendung für Simulation und Logik sinnvoll verwenden, weshalb die Berechnung von mehr Bildern als das Display darstellen kann verschwendete Rechenzeit ist (Davison, 2005).

#### 4.1.1.4 Aussehen

In der realen Welt kommt ein bestimmtes Aussehen durch Lichtreflexion zustande[9]. Das gilt auch für virtuelle Umgebungen, in denen Position und Farbe einer *Lichtquelle* das Aussehen der Objekte beeinflusst. Hauptsächlich aber wird das Aussehen hier durch Überlagerung der Flächen einer geometrischen Form mit Farben und Texturen bestimmt (Davison, 2005).

### 4.1.2 Simulation

> Rechnergestützte Simulation wird heute in unterschiedlichsten Anwendungen erfolgreich eingesetzt... und ganz allgemein [als] das Berechnen und Darstellen von Zeitverläufen bestimmter Größen in mathematischen Modellen bezeichnet, wie Ort, Kraft, demographische Daten, Kapitalflüsse, Aktienkurse, etc... (Preissler, 1995)

Die Simulation wird als das *Modell* hinter einer virtuellen Umgebung betrachtet. Das Modell besteht aus fundierten mathematischen Regeln und Gesetzen.

Ein solches Regelwerk, welches das Zusammenspiel zwischen Körpern und darauf einwirkenden Kräften beschreiben und berechnen kann bieten *Physics*- beziehungsweise *Dynamicsengines*.

[8] Das ist bei weitem nicht die einzige Möglichkeit, die Reifen zum Drehen zu bringen, nur die einfachste. Komplexer (und glaubwürdiger) wäre beispielsweise eine geschwindigkeitsabhängige, abspielbare Rotationsanimation oder die Verknüpfung mit einem Simulationskörper (in diesem Falle der Motor). Simulationskörper werden im Kapitel 4.1.2 behandelt.

[9] *http://de.wikipedia.org/w/index.php?title=Aussehen&oldid=45530006*

#### 4.1.2.1 Physikalische Eigenschaften:

Die physikalischen Eigenschaften einer Simulation basieren oft auf Modellen der klassischen Physik.

1. Die Umgebung enthält Körper und Auslöser. In der Physik ist ein Körper ein Objekt, das eine Masse hat und einen Raum einnimmt. In der klassischen Physik gilt: Wo ein Körper ist, kann kein anderer sein[10]. Körper können beispielsweise Hindernisse oder Vehikel sein. Auslöser sind zum Beispiel Spurhalter. Eine Kollision entsteht dann, wenn ein Körper einen anderen Körper oder Auslöser *berührt*.

2. Bestimmte globale *Parameter* haben Einfluss auf physikalische Körpereigenschaften. Ein Beispiel ist der Einfluss der Gravitationskonstante auf die Gewichtskraft[11].

#### 4.1.2.2 Aktive und passive Simulation:

Eine aktive Simulation läuft zeitgleich zum visuellen Geschehen ab. Ein Beispiel dafür ist virtuelles Dosenwerfen[12].

Bei der passiven Simulation werden Berechnungen vorgenommen, die erst im Nachhinein ausgewertet und visualisiert werden. Passive Simulationen sind meist rechenintensiver und hinterlassen einen enormen Datenberg zur Auswertung. Beispiele sind die Simulation der Klimaveränderung der letzten hundert Jahre oder die Kollision eines Wagens gegen eine Wand unter Einbezug diverser Fahrerassistenz-Systeme.

#### 4.1.2.3 Simulation Loop / Step

Während im *virtuellen* Universum Objekte durch Translation und Rotation manipuliert werden können, gilt es im *simulierten* Universum *Kräfte* auf die *Simulationskörper* der Teilnehmer wirken zu lassen. Das Resultat der Krafteinwirkung ist dann die neue Position und Orientierung es Körpers, die wir in der Renderloop nicht durch Translation oder Rotation sondern durch Überschreibung der Zustandsmatrix übernehmen (Millington, 2007).

#### 4.1.2.4 Simulationskörper

Virtuelle Objekte, die den Einflüssen einer Simulation ausgesetzt sein sollen, benötigen einen Simulationskörper Der Simulationskörper kann parallel in der virtuellen und der *simulierten Welt* existieren. Es besteht wie die visuelle Geometrie aus Partikeln, die mit speziellen *Joints* verknüpft werden. Die Formgeometrie des virtuellen Objektes kann als Basis für das

[10] http://de.wikipedia.org/w/index.php?title=K%C3%B6rper_%28Physik%29&oldid=42775028
[11] http://de.wikipedia.org/w/index.php?title=Gewichtskraft&oldid=44741543
[12] http://ieweb.etech.haw-hamburg.de/yamlt3/Virtual-Reality.86.0.html

*Gitter* des Simulationskörpers dienen. Das Gitter sollte gerade so groß sein, dass es um die Form eines Objekts herum gelegt werden kann. Ein Beispiel ist in Abbildung 4.2 zu sehen: das Gitter um die Sphäre stellt den Simulationskörper dar. Die Anzahl der Punkte im Gitter entspricht später der Genauigkeit, mit der Kräfte auf die umhüllte Form wirken können. Je mehr Punkte das Gitter hat, desto aufwendiger wird die Berechnung (Millington, 2007).

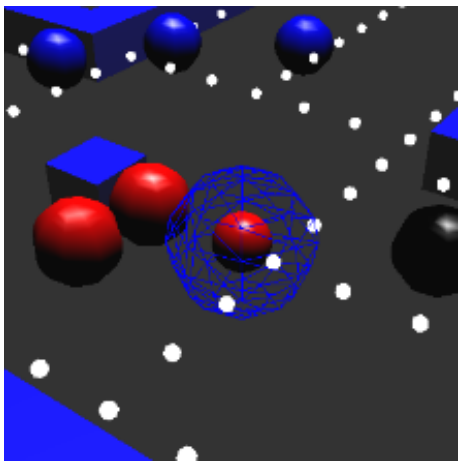

Abbildung 4.2: Form mit sichtbarem Simulationskörper (Gitter)

In einigen professionellen Anwendungen und Spielen werden Simulationskörper und Form so verknüpft, dass die Geometrie des Objekts sich entsprechend der Krafteinwirkung auf den Simulationskörper verändert oder gar zu Bruch geht. Diese Aspekte liegen jedoch außerhalb des Scopes dieser Arbeit.

## 4.2 Agenten und MAS

> Objects do it for free; agents do it because they want to (Wooldridge, 2002).

Agenten sind Computerprogramme die innerhalb einer Umgebung in der Lage sind, selbständig bestimmte Aktionen auszuführen. Sie verfolgen vorgegebene Ziele, denen sie reaktiv und proaktiv nachgehen. MAS steht für Mulitagentsysteme, in denen Agenten nach bestimmten sozialen Mustern kooperieren oder gegeneinander agieren können. Agenten müssen zur Erreichung ihrer Ziele mit vorkompilierten Plänen ausgestattet werden (Wooldridge, 2002).

## 4.3 Agent-Umgebung

Im nächsten Abschnitt übersetzen wir die Definition der Agent-Umgebung der JADEX Homepage[13] und untersuchen, welche Eigenschaften auf das Zusammenspiel eines Agenten mit einer virtuellen Umgebung zutreffen.

Agenten interagieren mit ihrer Umgebung auf unterschiedlichem Wege. Ein Agent observiert die Umgebung mit Sensoren und modifiziert sie mit Hilfe von Aktoren. Zusätzlich hat er die Möglichkeit, sich in seiner Umgebung zu bewegen. Diese Interaktionstypen ähneln denen eines Roboters in der realen Umgebung, können aber je nach Einsatz erheblich variieren.

(Russel und Norvig, 2003) klassifizieren die Eigenschaften einer Umgebung als:

Zugänglich / Unzugänglich: Zugängliche Umgebungen erlauben es Agenten, alle vorhandenen Informationen anzufordern. Reale Welten sind in diesem Sinne für Agenten unzugänglich, da es nicht möglich ist den globalen Zustand festzustellen.

Da unsere Umgebung der realen Welt nachgeahmt ist und der Agent nicht direkt, sondern über Vehikel mit dieser in Verbindung steht, gilt die Umgebung als unzugänglich.

Deterministisch / Nicht-Deterministisch: Ist eine Umgebung deterministisch, führen Aktionen eines Agenten garantiert zu einem bestimmten Effekt und vordefinierten globalen Status.

In der virtuellen Umgebung gehen die Aktionen mehrere Agenten ein. Die Simulation bestimmt das Ergebnis anhand der eingegangen Aktionen. Betrachten wir die Reihenfolge der Aktionen. Werden diese parallel und ungeordnet abgearbeitet, ist das Resultat im einzelnen nicht vorhersehbar und entsprechend nicht deterministisch. Kann für die Reihenfolge der Aktionen garantiert werden, lässt sich das selbe Resultat auch ein zweites mal produzieren, wenn keine zufälligen Aktionen statt gefunden haben.

Aus Agentsicht, bleibt die Umgebung aber weiterhin nicht deterministisch. Die gleiche Aktion kann in der Simulation unterschiedliche Auswirkungen haben.

Statisch / Dynamisch: Statische Umgebung ändern sich nur durch Aktionen eines Agenten. Dynamische Umgebungen verändern sich ohne Zutun eines Agenten.

Die virtuelle Umgebung kann auch von nicht Agenten geändert werden, demnach ist sie dynamisch.

Diskret / Kontinuierlich: Eine Umgebung ist dann diskret, wenn sich die Anzahl der möglichen Aktionen und Wahrnehmungen innerhalb der Umgebung zu einem bestimmten Zeitpunkt bestimmen lässt. Kontinuierlich ist eine Umgebung dann, wenn sich die

[13] http://vsis-www.informatik.uni-hamburg.de/projects/jadex/rm_environment.php

Anzahl möglicher Aktionen stetig ändert und sich die genaue Anzahl nur annähernd bestimmen lässt.

## 4.4 Vehikel

Vehikel sind per Definition Hilfsmittel; ein Mittel welches dazu dient, ein bestimmtes Ziel zu erreichen[14]. Betrachten wir das Vehikel zunächst als ein Transportmittel innerhalb einer Umgebung. Fahrer eines Vehikels können Anwender, Agenten oder eine beliebige Steuerungslogik sein.

## 4.5 Mixed-Reality

Der Begriff *Mixed-Reality* wird von (Milgram und Kishino, 1994) (zitiert in (Benford u. a., 1998)), als das Zusammenführen von realen und virtuellen Welten definiert, so dass Objekte aus der realen und virtuellen Welt in einem einzelnen Display präsent sind.

Geplant ist ein System zur Kopplung heterogener Systeme, wovon mindestens eines ein reales (mechanisches), von Software gesteuertes System ist, und ein anderes eine dynamische virtuelle Umgebung bereitstellt. Ansiedeln lässt sich dieses Konzept von der Mitte aus links des *Realitäts-Virtualitäts-Kontinuums* (Abbildung 4.3), in dem wir die Realität des mechanischen Systems um die Realität der virtuellen Umgebung erweitern.

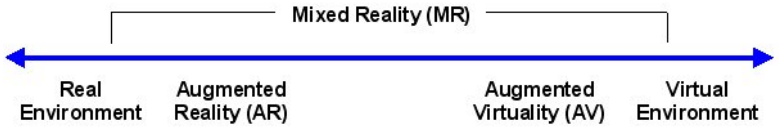


Abbildung 4.3: Realitäts-Virtualitäts-Kontinuum

[14] http://lexikon.meyers.de/meyers/Vehikel

# Kapitel 5

# Anwendungsfälle

Die Formulierung von Anwendungsfällen hilft dabei, sich in der noch frühen Phase des Vorhabens ein Bild von möglichen Einsatzgebieten der Software zu machen, ohne zu sehr ins Detail einer Implementierung zu gehen. Aus der Beschreibung der Anwendungsszenarien können Anforderungen an das System abgeleitet werden (Kahlbrandt, 2005).

## 5.1 Agent steuert virtuelles Objekt

**Akteure:** Entwickler (E), Agent (A), System (S)

1. E definiert einen Agenten A.
2. E erstellt Beschreibung der Umgebung.
3. S stellt eine virtuellen Umgebung bereit.
4. S generiert die Umgebung.
5. E kompiliert ein Plan für A, wie ein Objekt in der virtuellen Umgebung zu bewegen ist.
6. A führt den Plan aus
7. S informiert A über eine Kollision in der virtuellen Umgebung.

## 5.2 Agent steuert virtuelles Objekt (2)

### 5.2.1 Definition des Agenten und Gestaltung der Umgebung

**Akteure:** Entwickler (E), Agent (A), System (S)

1. E definiert[15] einen JADEX[16]agenten A.
2. E startet die von A benötigte Agent-Plattform.
   (Abbildung 5.1)
3. E entscheidet, wie die virtuelle Umgebung aussehen soll:
   (a) E entwirft eine Beschreibung der Umgebung mit Hilfe eines Editors.
   (b) Die Beschreibung wird auf das System S übertragen.
   (c) S visualisiert die Umgebung anhand der Beschreibung
   (Abbildung 5.2)

   oder

   (a) E verbindet sich über eine Clientsoftware mit der virtuellen Umgebung
   (b) E erstellt sich ein *Avatar*[17]
   (c) E gestaltet die Umgebung, mit den von der Clientsoftware bereitgestellten Werkzeugen
   (Abbildung 5.3)

### 5.2.2 Steuerung eines virtuellen Vehikels

**Akteure:** Entwickler (E), Agent (A), System (S)

1. E legt fest, welches Vehikel A in der Umgebung steuert. Das Vehikel stammt aus einer Bibliothek, die Vehikel für S oder auch andere System beinhaltet. Die virtuelle Umgebung soll von A als *Fahrer* des Vehikels erkundet. Das Vehikel bietet bestimmte Bewegungen innerhalb der Umgebung an. A verwendet ein festgelegtes Format zur Beschreibung von Bewegungsbefehlen. A weiß nicht, wie und ob das Vehikel eine Bewegung ausführt.
2. S zeigt das 3D-Modell eines Vehikels auf dem Bildschirm.
3. A bewegt das Vehikel.
4. S entscheidet ob die Bewegung eines Vehikels möglich ist:
   (a) A kollidiert durch die Bewegung mit einer Wand.

[15] Erstellung eines ADF (Agent-Definition-Files)
[16] http://vsis-www.informatik.uni-hamburg.de/projects/jadex/
[17] Ein Avatar ist ein virtueller Stellvertreter, dessen Aussehen der Entwickler festlegen kann

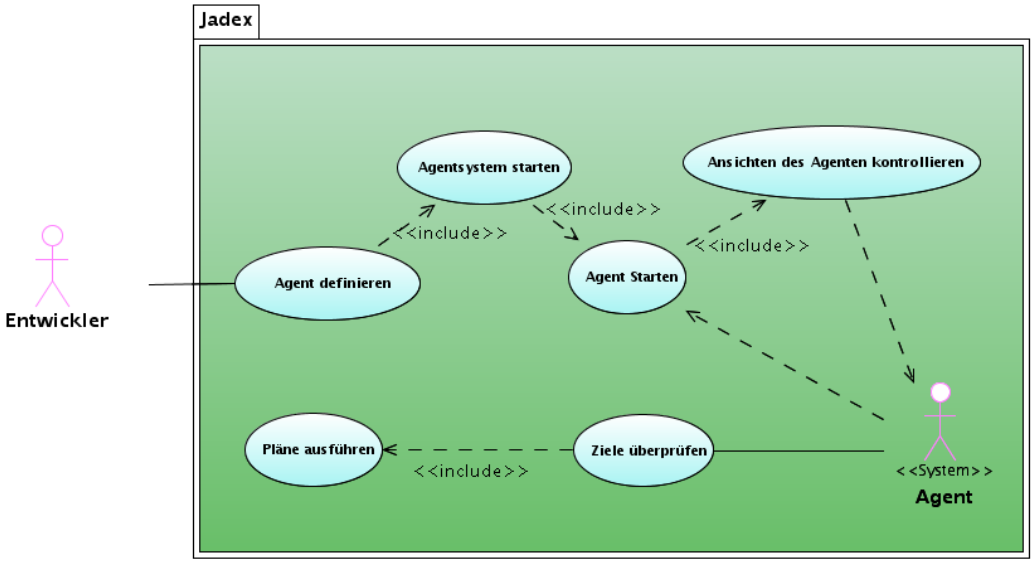


Abbildung 5.1: Definition eines Jadex-Agenten

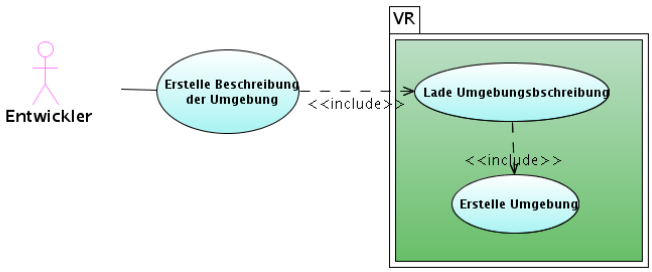


Abbildung 5.2: Gestaltung der Umgebung durch Beschreibung

(b) S für den Bewegungsablauf des Vehikels nur bis an die Grenze der Wand aus.

(c) Ein Kollisionsereignis wird von S registriert und an das von A gesteuerte Vehikel versendet

(d) A bekommt vom Vehikel die Rückmeldung, dass eine Kollision statt gefunden hat

oder

(a) S entscheidet die Bewegung war gültig und stellt die Bewegung dar

(Abbildung 5.4)

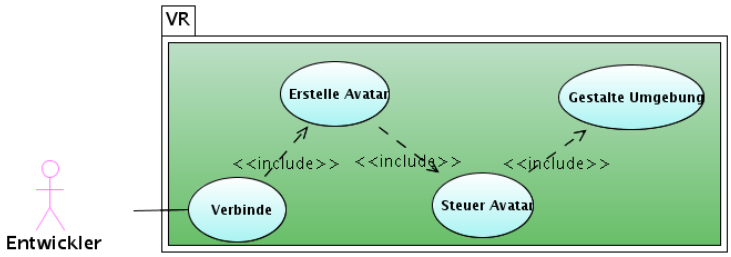


Abbildung 5.3: Immersive Gestaltung der Umgebung

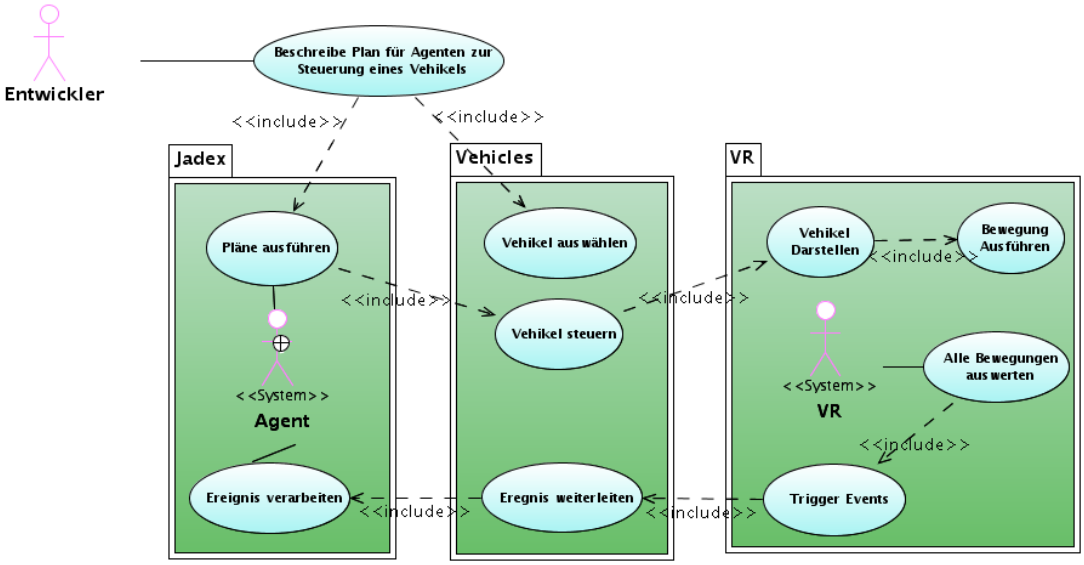


Abbildung 5.4: Jadex-Agent steuert Vehikel in der VR

### 5.2.3 Steuerung eines realen Vehikels

**Akteure:** Entwickler (E), Agent (A), System (S), Roboter (R), Trackingsystem (T)
Hinweis: R kann von einem Agenten gesteuert werden oder auch nicht. Wenn nicht, übernimmt die Rolle eines Agenten im Folgenden die Steuerungslogik von R

1. A unterscheidet R nicht von einem virtuellen Vehikel.
2. T meldet eine Attrappe von R in S an.
3. R führt ein Bewegungsbefehl von A aus.
4. S bekommt keine Bewegungsnachricht.

5. T liefert die jeweils aktuellen Positionsdaten von R an S.
6. S aktualisiert die Position der Attrappe von R
7. T liefert eine Position, wo in der virtuellen Welt aber eine Wand steht.
8. Ein Kollisionsereignis wird von S registriert und an R versendet
9. R informiert A über das Ereignis
10. S stellt die Attrappe gemäß der Position in der Wand dar

### 5.2.4 Anwendungsfelder

Ein System, welches die in den vorgehenden Anwendungsfall beschriebenen Funktionalitäten erfüllt, könnte auch für weitere Anwendungen eingesetzt werden. Darunter können wir uns folgende Beispiele vorstellen:

#### 5.2.4.1 Ein Mixed Reality Aufbau

Entscheidendes Merkmal eines Mixed Reality Aufbaus ist die Verbindung von virtuellen und physikalischen Komponenten[18] Unter anderem inspiriert von (Lindinger u. a., 2006) überlegen wir folgenden Aufbau:

Akteure: Entwickler (E), Agent (A), Roboter(R), System (S), Anwender (U)

Auf einem Tisch, dessen Oberfläche, einem Schachbrett ähnlich in Feldern aufgeteilt ist, wird eine virtuelle Umgebung projiziert. Auf dem Tisch liegen Figuren. Ein an S gekoppeltes Augmented-Reality-Tracking (kurz ART) (Hampshire u. a., 2006) System verfolgt die Figuren. Der Tisch befindet sich in einem CAVE[19]. An den Wänden des CAVE sieht U eine virtuelle Umgebung. Jede Figur auf dem Tisch wird auch als Objekt in der VR-Umgebung dargestellt. Eines der virtuellen Objekte stellt einen Roboter R dar, welcher sich außerhalb des CAVEs befinden. Der Entwickler E verbindet jedes weitere Objekt in der virtuellen Umgebung mit einem steuernden Agenten A. U verschiebt eine Figur auf dem Feld. Es gibt nun zwei Möglichkeiten:

1. Ist das der Figur zugeordnete Objekt rein virtuell, versucht A die nötigen Bewegungen auszuführen, um das Objekt in der virtuellen Umgebung an die gleiche Stelle zu bringen. Dabei unterliegt das Objekt den Einflüssen der virtuellen Umgebung, wie etwa Hindernissen, Schwerkraft und Steigungen. Kollidiert A zum Beispiel gegen virtuelle Wände, informiert S A über die Kollision.

[18] http://www.robocup-german-open.de/en/node/198

[19] http://de.wikipedia.org/w/index.php?title=Cave_Automatic_Virtual_Environment&oldid=51789006

2. Ist das Objekt mit R verknüpft, führt R die Bewegungen selbst aus. Der Roboter führt die Bewegung in der realen Umgebung aus. Er unterliegt dabei den physikalischen Einflüssen wie der Motorleistung oder einer rutschigen Unterlage. Ändert R seine Position, übergibt ein ART-System die neue Position an S, was die Position des virtuellen Stellvertreters aktualisiert. Kollidiert R zum Beispiel gegen virtuelle Wände, informiert S R über die Kollision. R ist so programmiert, dass er die Kollision für real hält, weshalb er einen neuen Weg sucht, die Position einzunehmen, die der Figur auf dem Tisch in der virtuellen Welt entspricht.

#### 5.2.4.2 Bälle einsammeln

Virtuelle Bälle werden in den Raum geschossen, ein Roboter soll die Bälle einsammeln. Dadurch soll die Planungsfähigkeit und mechanische Reaktionsgeschwindigkeit des Roboters getestet werden.

#### 5.2.4.3 Pendel balancieren

Ein Roboter balanciert ein virtuelles *tonnenschweres* Pendel. Dies wäre ein Fall zur Erprobung ausgefeilter Regelungstechnik.

#### 5.2.4.4 Verkehrssimulation

Ein autonomes Fahrzeug soll in einem virtuellen Verkehrsszenario die Verkehrsregeln einhalten. Es fährt dabei aber in einer leeren Halle. Die Sensoren zur Spureinhaltung und Abstandsmessung werden mit Informationen aus der virtuellen Umgebung gefüttert, damit das Vehikel auf Ereignisse in der simulierten Welt reagieren kann.

# Kapitel 6

# Anforderungen

Durch Formulierung früher Anforderungen zur Erfüllung der in den Anwendungsfällen beschriebenen Szenarien wird die initiale Richtung des Vorhabens vorgegeben. Die Anforderungen erheben wir in den Kategorien

*kann*: Die Implementierung würde im Idealfall diese Anforderung erfüllen, muss es aber nicht.

*soll*: Die Implementierung sollte die Anforderung erfüllen, muss es aber nicht.

*muss*: Die Implementierung muss diese Anforderung erfüllen, oder geändert werden bis sie es tut.

## 6.1 Fachliche Anforderungen

### 6.1.1 Dokumentation

Eine Dokumentation ist für die Nutzung und Weiterentwicklung des Systems durch Dritte notwendig. Die schriftliche Fassung dieser Arbeit *soll* auch als Dokumentation dienen. Der Quellcode *soll* zur besseren Verständlichkeit jeweils auf ein Teilgebiet bezogene Erläuterungen enthalten.

### 6.1.2 Trennung von Verantwortlichkeiten

Die Trennung der Verantwortlichkeiten *soll* es Entwicklern erlauben, sich auf das jeweilige Teilgebiet zu konzentrieren. Die Transparenz und Kohärenz des verteilten Systems *muss* dabei gewährleistet sein. Die frühestmögliche Trennung der Belange bei der Entwicklung eines

Systems ist die *Komponentisierung*. Eine Komponente [20] kapselt die innere Funktionalität von der Außenwelt ab. Ist die genaue Implementation der Komponente bekannt, spricht man von einer *Whitebox*. Sind die Schnittstellen, aber nicht die genaue Funktionsweise bekannt, wird sie als *Blackbox* bezeichnet. Eine Komponente *soll* austauschbar sein, ohne die Funktionalität des Systems zu beeinträchtigen.

Folgende Komponenten leiten wir aus dem Anwendungsfall 5.2.4.1 ab:

1. VR

   Aufgaben: Darstellung von Vehikeln, Erfassung von Ereignissen

2. Vehikel

   Aufgaben: Transportmittel der Agenten in der VR, Leiten Befehle an die VR und Ereignisse an den Fahrer zurück

3. Agenten

   Aufgaben: Steuerung virtueller oder realer Vehikel. Setzen Agentplattform / -Middleware voraus.

4. Roboter

   Aufgaben: Aus Sicht des Systems keine. Roboter gilt es in der VR abzubilden und über Ereignisse möglichst in Echtzeit zu informieren.

5. Positionsystem, ART

   Aufgaben: Versorgt die VR mit den aktuellen Positionsdaten der Roboter

### 6.1.3 Systemverantwortlichkeiten

Unter Systemverantwortlichkeiten verstehen wir die (Teil-)Aufgaben, für die das System zuständig ist (Kahlbrandt, 2005). Die frühe Eingrenzung der Verantwortlichkeiten teilt den Problembereich auf und erlaubt eine gezielte Suche nach der Lösung von Teilproblemen.

Das zu entwickelnde System versteht sich in erster Linie als Infrastruktur zur Kopplung heterogener Komponenten beziehungsweise. *unterschiedlicher Realitäten*. Zu den Teilaufgaben gehören:

1. Integration der Komponenten durch den Einsatz geeigneter *Konnektoren*[21]

[20] Eine Komponente ist eine modulare Einheit mit wohldefinierten erforderlichen und bereitgestellten Schnittstellen, die in ihrer Umgebung ersetzbar ist. (OMG,2004b) in (Tanenbaum und van Steen, 2008)

[21] *Konnektor:* Mechanismus, der Kommunikation, Koordination oder Kooperation zwischen Komponenten vermittelt (Mehta et al., 2000, und Shaw und Clements, 1997). Ein Konnektor kann beispielsweise aus den Einrichtungen für (entfernte) Prozeduraufrufe, für die Weitergabe von Nachrichten oder für Datenströme gebildet werden (Tanenbaum und van Steen, 2008)

2. Kontrolle eines virtuellen Objekts ermöglichen.

3. Erfüllung weicher Echtzeitanforderungen bei der Aktualisierung eines virtuellen Objekts durch ein ART oder die Robotersteuerungslogik.

4. Auslieferung virtueller Ereignisse an alle Teilnehmer.

## 6.2 Technische Anforderungen

### 6.2.1 Virtual Reality

Die Virtualisierung einer Umgebung übernehmen eine Visualisierungs- und eine Simulationskomponente.

#### 6.2.1.1 Visualisierung

Ein Aufgabe der Visualisierung ist, Objekte in der virtuellen Umgebung aus mehreren Perspektiven darzustellen. Die Projektion sollte auf einem einfachen Computermonitor und optional in einem CAVE erfolgen. Als Komponente sollte die Visualisierung austauschbar bleiben, inbesondere weil die Engines jeweils auf bestimmte Visualisierungsaspekte fokusiert sind. Ein externer Vergleich[22] veranschaulicht die Unterschiede.

1. Die Visualisierungslösung *muss* OpenGL unterstützen, *kann* DirectX unterstützen. Die Plattformbindung von DirectX ist ein großer Nachteil bei der Entwicklung des Prototyps, weil sie zum größten Teil auf Linux Distributionen[23] erfolgt.

2. Es *soll* in der Visualisierungslösung möglich sein, gleichzeitig mehrere Kameraperspektiven auf eine Szene in der virtuellen Welt zu haben: Eine Übersicht für den Leitstand der Simulation und die Verfolgersicht für Teilnehmer der Simulation.

3. Die Darstellung komplexer 3D-Szenen ist äußerst rechenintensiv. Ein Mechanismus zur Verteilung der Renderingbefehle *kann* die Berechnungszeit verkürzen.

4. Es *soll* mindestens einem Betrachter möglich sein, mit der Szene zu interagieren. Beispiele für Interaktion gehen vom *Highlighting* von Objekten bis hin zur Kontrolle *(Drag'n'Drop)* mit Hilfe von herkömmlichen Eingabegeräten wie Maus und Tastatur.

5. Die Lösung *soll* sich zur Erstellung und einfachen Anbindung eines Umgebungsprogramms für Jadex Agenten eignen. JADEX Agenten führen Pläne in Java Code aus, weshalb eine Java Lösung bevorzugt werden sollte.

[22] http://aviatrix3d.j3d.org/comparison.html
[23] http://de.opensuse.org/

#### 6.2.1.2 Simulation

Die Aufgabe der Simulation ist es exakte Berechnungen zum Zusammenspiel aller am Modell beteiligten Objekten durchzuführen und mögliche Ereignisse aus den Resultaten abzuleiten.

1. Der Systembetrieb *muss* von der Simulationskomponente unabhängig sein.
2. Damit die Simulation exakte Berechnungen ausführen kann, *muss* sichergestellt sein das alle Vorgänge für die Auswertung gespeichert werden.
3. Das Modell *soll* sich durch komplexere Modelle beliebig austauschen lassen, ohne die Funktionalität des Gesamtsystems zu beeinträchtigen.
4. Ein virtuelles Objekt *soll* wahlweise an der Simulation teilnehmen oder nicht.
5. Die Simulation *soll* Kollisionsereignisse zwischen zwei oder mehr Objekten erfassen.

### 6.2.2 Kommunikation

Durch Kommunikation wird der Zusammenhang zwischen den Komponenten hergestellt.

1. Kommunikation, die die Simulationskomponente mit einbezieht *muss* persistent erfolgen.
2. Ein Agent *muss* unabhängig von der Bearbeitungsdauer einer Nachricht durch das System anderen Aufgaben nachgehen können.
3. Die Kopplung der Komponenten durch Konnektoren *soll* möglichst lose erfolgen.
4. Für Roboter und Tracking-Systeme *soll* das System die maximale Bearbeitungszeit einer Anfrage garantieren.
5. Das zu verwendende Protokoll *soll* auf TCP/IP[24] Netzwerke aufbauen.

[24] http://www.ietf.org/rfc/rfc0675.txt

# Kapitel 7

# Analyse

In diesem Kapitel wird die Problemstellung aus Kapitel 2.4 behandelt und Lösungsansätze diskutiert.

Es wird in fachlicher und technischer Analyse unterteilt. Die fachliche Analyse bezieht sich auf die inhaltliche Auseinandersetzung mit der Problematik. Nach der Erfassung von Grundlegenden Zusammenhängen werden vorhandene Lösungsmuster und deren Anwendbarkeit auf die Teilaufgaben untersucht.

Die technische Analyse setzt sich mit systembedingten Voraussetzungen für die Machbarkeit einer Lösung auseinander. Existierende Technologien werden untersucht und deren Einsatz diskutiert. Abgeschlossen wird das Kapitel mit einer Aufstellung konkretisierter Anforderungen an das System sowie einer tabellarischen Auswertung der untersuchten Technologien.

## 7.1 Fachliche Analyse

### 7.1.1 Kameraperspektive

Folgende Perspektiven sind in den Anforderungen enthalten:

**Übersichtsperspektive** Sie ermöglicht dem Beobachter den Überblick über das gesamte Geschehen der Simulation. Optimaler weise ist diese Kamera zur Laufzeit frei positionierbar.

**Verfolgerkamera** Diese Kamera heftet sich an ein Vehikel und zeigt die virtuelle Welt aus Sicht des Fahrers.

Zur Darstellung des Geschehens in der virtuellen Umgebung wird mindestens die Übersichtsperspektive benötigt. Die Verfolgerkamera setzt zunächst voraus, dass wir ein Objekt in der virtuellen Simulation gezielt auswählen können. Das hängt von der Realisierung einer Schnittstelle für Interaktionen ab.

### 7.1.2 Interaktion

Die Visualisierungskomponente kann, muss aber nicht die erste Anlaufstelle für interaktive Ereignisse sein. Betrachten wir folgende Fälle:

1. Die Visualisierungslösung bietet einen Input-Handler als interaktive Schnittstelle an, der die Bewegungen der *Mouse* registriert. Erfolgt eine Auswahl, wird ein *unsichtbarer* Strahl orthogonal zur Koordinate unter dem *Cursor* durch den virtuellen Raum erzeugt. Das erste *sichtbare* Objekt, welches vom Strahl getroffen wird, ist das vom Anwender ausgewählte (Davison, 2005). Das ist zu diesem Zeitpunkt keiner Komponente außer der Visualisierung bekannt.

2. Die Bewegungen des Anwenders werden von einem externen ART System registriert. Die Zuordnung zu einem Objekt in der virtuellen Welt erfolgt erst im Nachhinein durch Überlagerung der Koordinatenräume.

Diese beiden Fälle sollen verdeutlichen, wie die Visualisierung sowohl direkt als auch indirekt an der Verarbeitung interaktiver Eingaben beteiligt sein kann.

### 7.1.3 Bewegung

Die VR erhält von außen Anweisungen, Objekte innerhalb der virtuellen Welt zu verschieben, zu rotieren oder die Position schlagartig zu ändern.

Eine Anweisung könnte lauten *rotiere Objekt 3 Einheiten nach links und 5 Einheiten nach vorne* oder *setze Position von Objekt = Neue Position*. Die VR hat dafür zu sorgen, die Anweisung auf das virtuelle Objekt zu übertragen. Ebenfalls muss die Komponente in der Lage sein, jederzeit die Position und Orientierung eines Objekts in der virtuellen Umgebung mitzuteilen.

**Mögliche Bewegungen** Alle Bewegungen, die ein virtuelles Objekt ausführen kann lassen sich durch Positions- und Richtungsänderung, beziehungsweise Translation und Rotation eines Objektes im drei-dimensionalen (3D) Raum ausdrücken. Einzelne Bewegungen beziehen sich immer auf jeweils ein bestimmtes Objekt, und können ausgehend von der Blickrichtung desselben angewendet werden. Somit lassen sich Bewegungen im 3D Raum auf folgende Beschreibung übertragen.

```
 object rotate
        (around) X|Y|Z (axis)
(anti-) clockwise
(by) w unit
```

```
 object move
(in) X|Y|Z (direction)
(by) n unit
```

Diese Beschreibung soll insbesondere für Agenten so umgesetzt werden, dass sie die einheitliche Steuerung unterschiedlicher Vehikel ermöglicht.

Im Falle von Robotern ist es wenig sinnvoll Bewegungsabläufe virtuell nachzumachen. Ein einfaches Beispiel: Der Roboter beschleunigt, rutscht aber aus. Externe Positionierungs- und ART Systeme haben sich zur Bestimmung der Position von Objekten im Raum etabliert. Ein solches Systeme implementierte bereits (Burka, 2007). Daher nehmen wir in die Liste möglicher Anweisungen mit auf:

```
object setposition (=) (x,y,z)
object lookat (=) (x,y,z)
```

### 7.1.4 Vehikel

Vehikel können in der virtuellen oder in der realen Umgebung situiert sein. Es wird zwischen Vehikel als virtuelle Roboter, Roboter als reale Vehikel, und Mockups der Roboter in der virtuellen Umgebung unterschieden. Wichtig für Agenten ist, dass sich alle Vehikel auf gleiche Weiße steuern lassen.

**Virtuelle Vehikel** bestehen aus einer geometrischen Form, einem Simulationskörper und haben ein Aussehen.

Sie können in der virtuellen Umgebung bestimmte Bewegungen selbständig ausführen. Ein Simulationsmodell (Kap. 4.1.2) könnte für virtuelle Vehikel zusätzlich festlegen, dass sie sich nicht durch andere Körper hindurch bewegen können.

Virtuelle Vehikel sollen Agenten die Erkundung der virtuellen Welt ermöglichen. Die Anweisungen

```
object setposition (=) (x,y,z)
object lookat (=) (x,y,z)
```

gelten nicht für virtuelle Vehikel. Sie würden die Autonomie des Agenten hintergehen.

Für Agenten muss das virtuelle Vehikel *transportabel* sein, damit sie nicht an Mobilität verlieren. Das heißt im Falle von Jadex Agenten, dass es serialisierbar sein muss. Die Bewegungs- und Transportabilitätsaspekte des Vehikel lassen sich als eine Art *Fernbedienung* abstrahieren. Die Fernbedienung als Steuerungsschnittstelle des Vehikels ist für den Agenten transportabler als das gesamte Vehikel. Diese Abstraktion verbietet dem Agenten allerdings das Aussehen und Faktoren wie das Gewicht des Vehikels zu beeinflussen, was wir als hinnehmbar betrachten.

**Roboter** sind reale Vehikel, die mit Hilfe eines virtuellen Mockups an der VR teilnehmen. Die Position und Orientierung des Roboters wird der VR von externen Tracking-System vorgegeben. Diese Daten werden auf das Mockups übertragen. Ein Roboter, der die Ereignisse der virtuellen Umgebung mitbekommen und verarbeiten soll, muss gemäß den technischen Anforderungen (Kap. 6.2.2) über TCP/IP Konnektivität verfügen. Ein Beispiel ist Robotino[25]. Andernfalls kann das Abholen eines ein externes System übernehmen, welches mit dem Roboter kommunizieren kann.

Im Einzelfall können Roboter von Agenten gesteuert werden. Nützlicherweise würde das Steuerungssystem des Roboters die gleiche Schnittstelle wie die der virtuellen Vehikel implementieren, damit es für den Agenten keinen Unterschied macht, ob sie virtuelle oder reale Vehikel steuern. Das System mischt sich bei der direkten Steuerung der Roboter durch den Agenten aber nicht weiter ein.

**Mockups** (Attrappen) stellen die räumliche Präsenz eines Roboters in der VR Umgebung dar. Sie sind keine akkuraten Kopien von Robotern. Durch das Mockup soll die VR die Lage und Orientierung eines Roboters in der virtuellen Umgebung bei der Berechnung des nächsten Simulationsschrittes mit einbeziehen können. Mockups sind den Effekten der virtuellen Simulation allerdings nicht ausgesetzt.

### 7.1.5 Kommunikation

Die Kommunikation zwischen den Komponenten bildet das Herzstück eines verteilten Systems. Die bekanntesten

> Middleware-Systeme, die die Kommunikation zwischen Prozessen durch Abstraktion der Transportebene vereinfachen (Tanenbaum und van Steen, 2008),

auch *Middleware-Kommunikationsprotokolle* genannt, sind:

- Entfernter Prozeduraufruf (Remote Procedure Call, RPC)
  Beispiele:
  - Remote Method Invocation (RMI[26])
  - Common Object Request Broker Architecture (CORBA[27])
- Nachrichtenorientierte Kommunikation
  Beispiele:

[25] http://www.festo-didactic.com/de-de/lernsysteme/neu-robotino-lernen-mit-robotern/
[26] http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp
[27] http://www.omg.org/corba/

  - Java Messaging System (JMS[28])
  - Store And Forward Message Queue (SAFMQ[29]

- Streamorientierte Kommunikation
  Beispiel:
  - Real Time Streaming Protocol (RTSP[30])
- Multicasting
  Beispiel:
  - Edutella Peer-to-Peer (P2P[31])

Die losesten Form der Kopplung ist der reine Datenaustausch zwischen zwei Modulen (Raasch, 1991). In der Objektorierentierung ist auch von Nachrichtenaustausch die Rede. Versteht ein Objekt die Nachricht, werden die darin enthaltenen Daten in einer Methode verarbeitet. Andernfalls ignoriert es die Nachricht oder gibt einen Fehler zurück.

Betrachten wir nachrichtenorientierte Kommunikation, anhand des einfachen Beispiels: Sender *A* schickt Empfänger *B* eine Nachricht, bedeutet lose gekoppelt, dass

- *A* nicht wissen muss, wie *B* die Nachricht genau bekommt.

  Nachrichten zwischen einer Gruppe von Computern / Prozessen durchlaufen mehrere Schichten. Jede Schicht behandelt einen bestimmten Aspekt der Kommunikation, ohne zu wissen, was die Schicht darunter macht. Zwischen den Schichten existieren fest definierte Schnittstellen. [32] Wichtig für uns ist die Schicht zwischen Anwendung und Transport. Dies ist das Middleware-Kommunikationsprotokoll. Wie die Middleware eine Nachricht genau überträgt, erfährt der Sender nicht. Es wird aber eine Schnittstelle benötigt.

- *A* nicht wissen muss, was *B* mit der Nachricht macht.

  Der Inhalt einer Nachricht ist beliebig. A unternimmt den Versuch Daten anzuliefern. B kann A ein Resultat zurück geben, muss es aber nicht.

- *A* nicht wissen muss, wann *B* die Nachricht liest.

  Sind A und B zeitlich entkoppelt, tauschen Sie Nachrichten aus ohne zu Wissen ob der andere aktiv ist (Abbildung 7.1)

[28] http://java.sun.com/products/jms/
[29] http://safmq.sourceforge.net/
[30] http://tools.ietf.org/html/rfc2326
[31] http://www.edutella.org/edutella.shtml
[32] Die einzelnen Schichten einer Nachrichtenübertragung werden im OSI-Modell veranschaulicht; ein für die Netzwerkkommunikation angepasstes Referenzmodell findet sich in (Tanenbaum und van Steen, 2008, Abbildung 4.3)

- *A* sicher gehen kann, dass *B* die Nachricht bekommt.

  Andernfalls sind die Systeme nicht gekoppelt.

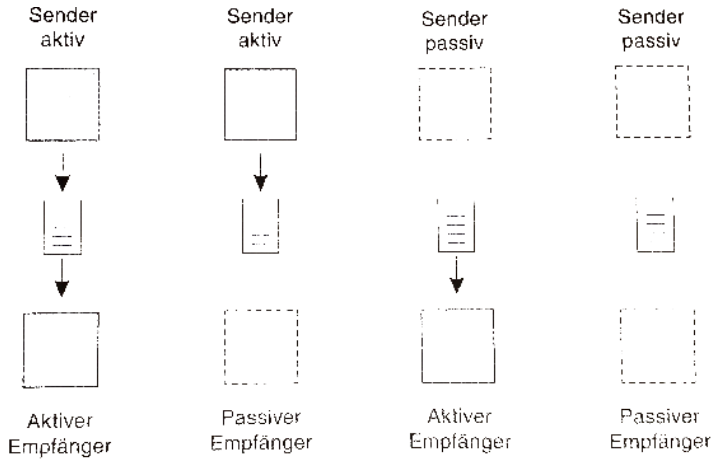


Abbildung 7.1: Vier Kombinationen für lose gekoppelte Kommunikation unter Verwendung von Warteschlangen (Tanenbaum und van Steen, 2008, Abb. 4.17)

#### 7.1.5.1 Empfänger und Sender

Bevor wir die Aspekte und Modelle nachrichtenorientierter Kommunikation genauer untersuchen, ist es sinnvoll die Komponente nach ihrem Kommunikationsbedürfnis als Empfänger und/oder Sender von Nachrichten einzuordnen:

**Die Fernbedienung** agiert als Sender von Befehlen an Vehikel.

**Das Vehikel** ist Empfänger von Befehlen einer Fernbedienung.

**Die Visualisierungskomponente** leitet Befehle an die virtuelle Objekte weiter, wenn diese die Transformation selbst ausführen können, andernfalls erledigt die Komponente die Transformation.

**Die Simulationskomponente** ist Empfänger und Sender. Sie empfängt ebenfalls Nachrichten, die Befehle an die virtuellen Vehikel darstellen. Ist ein Simulationsmodul vorhanden, steht es in der Verarbeitungshierarchie allerdings über der Visualisierung. Die Simulation verarbeitet die Befehle und sendet kontinuierlich den aktuellen Zustand der Objekte an das Visualisierungsmodul. Während eines Simulationsschritts können Ereignisse erfasst werden, die an die beteiligten Vehikel gesendet werden.

**VR** Visualisierung und Simulation interessieren sich beide für Befehle an virtuelle Objekte. Ist die Simulationskomponente vorhanden, muss es die Nachrichten *vor* der Visualisierung bekommen. Dieses Koordinationsproblem lässt sich durch Vereinigung der Komponenten zur VR Komponente aus der nachrichtenorientierten Kommunikationsebene auslagern (und dadurch einfacher lösen), was die VR zum alleinigen, Empfänger von Befehlen und Sender von Ereignissen macht.

#### 7.1.5.2 Zeitliche Koppelung

Sie beschreibt im wesentlichen ob beim Versand einer Nachricht der Empfänger aktiv sein muss oder nicht (Tanenbaum und van Steen, 2008). Dazu werden folgende Fragen an das zu entwickelnde System gestellt:

1. Sind die Nachrichten persistent (dauerhaft) oder flüchtig?
2. Läuft der Nachrichtenaustausch synchron (zeitgleich) oder asynchron ab?

- Prozesse, die persistente Nachrichten asychron austauschen sind *zeitlich entkoppelt*.
- Prozesse, die flüchtige Nachrichten synchron austauschen sind *zeitlich gekoppelt*.
- Prozesse, die flüchtige Nachrichten asychron austauschen sind beim Austausch *zeitlich ent-*, bei der Bearbeitungszeit *gekoppelt*.
- Prozesse, die persistente Nachrichten synchron austauschen sind beim Austausch *zeitlich ge-*, bei der Bearbeitungszeit *entkoppelt*.

Wir beantworten diese Fragen, in dem wir den Kommunikationsbedarf der Teilnehmer einer passiven und einer aktiven VR Simulation untersuchen:

#### 7.1.5.3 Kommunikationsbedarf bei passiver Simulation

**Die Agenten** steuern virtuelle Vehikel. Die Fernbedienung unterliegt allein der Kontrolle des Agenten. Die Kommunikation zwischen Agenten und Fernbedienung erfolgt synchron und flüchtig.

**Die Fernbedienung** übernimmt die Weiterleitung von Befehle an das Vehikel. Diese befindet sich bei virtuellen Vehikel innerhalb der VR Umgebung. Da die Simulation passiv erfolgt, muss das System Nachrichten an das Vehikel in der Reihenfolge des Eintreffens zwischenspeichern. Die Kommunikation zwischen Fernbedienung und virtuellem Vehikel erfolgt zeitlich entkoppelt.

**Ereignisse** entstehen in der VR Umgebung und werden an virtuelle Vehikel und Roboter weitergeleitet. Die Ereignismeldungen sollten bis zur Abholung zwischengespeichert werden. Die Kommunikation erfolgt demnach asynchron und persistent.

**Trackingsysteme** teilen der Umgebung die aktuelle Position der Roboter mit. In einem Szenario passiver Simulation reicht es aus, wenn die Infrastruktur die Nachrichten in der Ankunftsreihenfolge speichert, so dass die VR im Nachhinein die Fahrt der Roboter darstellen kann.

#### 7.1.5.4 Kommunikationsbedarf bei aktiver Simulation

Wird eine aktive Simulation betrieben, müssen eintreffende Nachrichten möglichst in Echtzeit bearbeitet werden. Das Geschehen in der Simulation soll mit der Realität der Teilnehmer zeitlich übereinstimmen. Wird die Simulation überfordert, dürfen Nachrichten nicht verloren gehen, weshalb sie auch im aktiven Simulationsbetrieb persistent bleiben. Das hat unweigerlich zur Folge, dass die Simulation unter bestimmten Umständen nicht synchron zur Zeit der Teilnehmer ausgeführt werden kann.

Dies kann für die Teilnehmer akzeptabel sein oder nicht. Damit ein Teilnehmer herausfinden kann ob das System seine Nachricht innerhalb einer bestimmten Zeit abgearbeitet hat, versieht er die Nachricht mit einer TTL (Time-To-Live). Nach Ablauf dieser Zeit muss die Simulation die Nachricht des Teilnehmers abgeholt oder das System einen Fehler an den Teilnehmer gemeldet haben.

#### 7.1.5.5 Message-Queues

Aufgrund der definierten Anforderung persistenter Nachrichten betrachten wir Message-Queues als ein geeignetes Mittel für den Nachrichtenaustausch. Message-Queue Systeme basieren auf dem *Client/Server Modell*: Der Server stellt eine Message-Queue zur Verfügung auf der Clients Nachrichten abholen und deponieren können. Klassisch sind Message-Queue Implementierungen für Anwendungen gedacht, bei denen mehrere Sekunden zwischen Versand und Auswertung der Nachricht vergehen können (Tanenbaum und van Steen, 2008). Sie eignen sich in unserem Sinne dann hauptsächlich für die Ausführung einer lose gekoppelten, passiven Simulation. Ob sich Message-Queues auch für den Betrieb einer aktiven Simulation eignen, sollen Performancemessungen (9.1) zeigen.

#### 7.1.5.6 Zeit

Der Nachrichtenaustausch zwischen VR und Vehikel ist zunächst zeitlich entkoppelt. Der zentrale Taktgeber ist die VR, die die Nachrichten abholt und bei deren Verarbeitung Ereignisse erzeugen kann. Eine Ausnahme besteht für Systeme, die weiche Echtzeitanforde-

rungen stellen. In diesem Fall besteht zeitliche Koppelung. Diese können bereits die maximale Bearbeitungszeit einer Nachricht definieren. Läuft diese Zeit ab, sollten sie eine Fehlermeldung erhalten. Die verlässlichste Zeitmessung ist hier vom Eintritt der Nachricht in die Message-Queue der VR bis zur Bearbeitung der Nachricht, da nur der Zeitstempel des Message-Queue Servers eine Rolle spielt. Für weiche Echtzeitanforderungen ist es ausreichend, wenn sich die die voraussichtliche Bearbeitungszeit einer Nachricht annähern lässt.

#### 7.1.5.7 Referenzielle Koppelung

Referenzielle Koppelung bezieht sich auf das Wissen über die Identität des Empfängers beim Versand einer Nachricht. Bekannte Muster referenziell gekoppelter Kommunikation sind das *Mailbox*- und das *Blackboardmodell*. Dagegen gilt das *Publish/Subscribe* Modell als referenziell entkoppelt. Bei der Beschreibung der folgenden Modelle wird auf (Tanenbaum und van Steen, 2008) und (Grand, 2002) Bezug genommen.

#### 7.1.5.8 Mailbox

Das Mailbox Modell (auch Point-to-Point genannt) entspricht der Kommunikation über Post oder Email. Der Versender der Nachricht bestimmt das Thema der Nachricht und die Empfänger, wozu er erst deren Adresse erfahren muss. Durch die Verwendung ausschließlich bekannter Adressen ist die Kommunikation referenziell gekoppelt.

#### 7.1.5.9 Blackboard

Das Blackboard Modell entspricht der Tafel in einem Klassenraum oder der Kommunikation in Newsgroups. Darin wird eine Nachricht hinterlegt die jedem zugänglich ist. Mehrere Blackboards werden nach Themengebiet unterteilt. Die Kommunikation über ein Blackboard ist referenziell gekoppelt.

#### 7.1.5.10 Publish/Subscribe

In diesem Modell übernimmt die referenzielle Zuordnung ein *Nachrichten-Broker*. Die Hauptaufgabe des Brokers (Vermittlers) besteht darin, Nachrichten die einem bestimmten Thema zugeordnet sind an Empfänger zu übermitteln, die sich für diesen Inhalt interessieren. Der Nachrichten-Broker wird als Gateway (Zugangssystem) auf Anwendungsebene realisiert. Der Publisher erzeugt Nachrichten zu bestimmten Themen. Die Subscriber registrieren ihr Interesse für bestimmte Themen beim Broker, der ihnen passende Nachrichten in der Reihenfolge der Ankunft ausliefert. Das Modell gilt als referenziell entkoppelt, da Publisher und Subscriber nicht von einander wissen.

Eine erweiterte Aufgabe des Brokers besteht darin für die Umwandlung der Nachricht zu sorgen, falls der Empfänger das Nachrichtenformat des Senders nicht versteht. Diese Umwandlungsaufgabe stellen wir aber zunächst in den Hintergrund[33].

Die referenzielle Entkoppelung durch den Einsatz eines Nachrichten-Brokers stellt auch ein Problem dar: Der Broker ist die zentrale Anlaufstelle für alle Nachrichten. Dies kann sich als Nadelöhr für das System herausstellen und das komplette System zum Stillstand bringen, wenn es ausfällt. Da es sich bei den Implementierungen nachrichtenorientierter Kommunikation meist um Client-Server basierte Anwendungen handelt, ist die Skalierbarkeit und Ausfallsicherheit schwerer zu gewährleisten als bei dezentralen Ansätzen wir Peer-to-Peer.

#### 7.1.5.11 Eignung der Modelle

Durch die Vereinigung von Visualisierung und Simulation zur VR Komponente stellen wir fest, dass nur ein Empfänger für Vehikelnachrichten vorhanden ist. Es wir nur eine Steuerungs-Message-Queue benötigt.

Anders die Verteilung von Ereignisnachrichten, für die sich mehrere Empfänger interessieren können. Wir gehen auch in diesem Fall von der garantierten Auslieferung einer Nachricht, über eine Ereignis-Message-Queue aus.

Für den Versand von Ereignissen an einen oder mehrere Empfänger sind folgende Verteilungsverfahren vorstellbar:

1. Die Nachricht wird über eine globale Message-Queue versendet
   - Alle Empfänger lesen alle Ereignisnachrichten
2. Die Nachricht wird über eine globale Message-Queue versendet und hat ein Thema
   - Nur Empfänger die sich für das Thema interessieren lesen die Nachricht
3. Eine Kopie der Nachricht wird an jeden beteiligten Empfänger, über eine gemeinsame Message-Queue versendet
   - Der Empfänger kann sich eine Message-Queue mit anderen Empfänger teilen
4. Eine Kopie der Nachricht wird an jeden beteiligten Empfänger, an seine Message-Queue versendet
   - Der Empfänger verfügt über eine eigene Message-Queue

[33] Dazu fehlt uns die Spezifikation einer Nachricht.

#### Das erste Verfahren

Das Verfahren eignet sich nicht für Message-Queues, in denen eine Nachricht nach ihrer Abholung aus der Queue entfernt wird (SAFMQ). Es müssten alle Empfänger erst alle Nachrichten lesen um zu bestimmen, ob die Nachricht für Sie relevant ist.

#### Das zweite Verfahren

Setzt entweder das Publish/Subscribe oder das Blackboard Modell voraus: Nur Empfänger die sich für das Thema interessieren lesen die Nachricht. Das Thema dürfte aber Beispielsweise nicht *Ereignis=Kollision* lauten, da sich hierfür potentiell alle Empfänger interessieren. Ein Broker wäre notwendig, der die Nachricht *Ereignis und beteiligte Objekte* untersucht und nur an Empfänger weiter gibt, die sich für diese Ereignis und dem Objekt registriert haben.

#### Das dritte Verfahren

Stellt den größten Aufwand für den Sender von Ereignissen dar, da er die selbe Nachricht mehrmals versenden muss. Auch ist der Speicherbedarf für Nachrichten am höchsten. Ein Broker wird nicht benötigt. Der Sender muss zusätzlich jede Nachricht kennzeichnen können. Wenn ein Empfänger die Ereignis-Message-Queue mit anderen Empfängern teilt, sollte er darin nur für ihn bestimmte Ereignisse abholen. Es hindert den Empfänger aber zunächst nichts daran, Nachrichten abzuholen die nicht für ihn bestimmt sind. Kann man nicht von einem verantwortungsvollen Umgang der Empfänger ausgehen, sollte das System dem Empfänger ermöglichen die Adresse der Ereignis-Message-Queue für den Empfang ihn bestimmter Ereignismeldungen selbst vorzugeben.

#### Das vierte Verfahren

Entspricht weitestgehend dem Mailboxmodell. Es Benötigt den selben Aufwand und hohen Speicherbedarf wie das dritte Verfahren. Nur statt Nachrichten zu kennzeichnen, muss der Sender verlässlich jeden einzelnen Empfänger adressieren können.

### 7.1.5.12 Beurteilung

Das erste Verfahren ist sehr ineffizient, da alle Empfänger alle Nachrichten lesen würden. Das zweite Verfahren setzt das Publish/Subscribe Modell und den Einsatz eines Brokers voraus. Als Blackboard Modell ist es nicht realisierbar, da pro Thema eine Message-Queue benötigt würde es aber zu viele Themenvariationen gibt.

Das dritte und vierte Verfahren benötigen keinen Broker. Das dritte hat gegenüber dem vierten den Vorteil, das System mit nur zwei Message-Queue betreiben zu können. Die Ver-

fahren lassen sich kombinieren, wenn der Empfänger entscheiden kann wohin das Ereignis gehen soll. Bei beiden besteht referenzielle Koppelung.

Die Skalierbarkeit des Brokers könnte ein Problem darstellen, da es eine zentrale Instanz gäbe die alle Nachrichten bearbeiten müsste. Die Arbeitsverteilung im vierten Verfahren ist besser, da es jedem Empfänger ermöglicht eine eigene Ereignis-Message-Queue zu betreiben und nur eigene Nachrichten zur bearbeiten. Idealerweise werden Empfänger auf dem Server ausgeführt, wo sie auch ihre eigene Message-Queue haben. Wenn das Senden einer Nachricht pro beteiligter Empfänger die VR nicht überfordert, ist die Kombination vom dritten und vierten Verfahren die bessere Wahl.

#### 7.1.5.13 Struktur einer Nachricht

Wir orientieren uns bei der Struktur einer Nachricht an eine email. Sie hat ein Thema (Subject oder Label) und ein Inhalt (Body). Das Label sagt uns worum es im Body geht.

#### 7.1.5.14 Spezifikation einer Nachricht

Betrachten wir die einzelnen Komponenten des Systems als Module, lässt sich aus der SD-Modellbewertung schliessen, dass die loseste Form der Kopplung die normale Datenkopplung ist (Raasch, 1991). Als Übergabeparameter sind nur Objekte von einfachen Datentypen (einzelnes Feld oder homogene Tabelle) erlaubt.

Einfache Datenkopplung schreibt vor das innerhalb einer Nachricht nur einheitliche Datentypen verwendet werden. Das ist ein Problem wenn wir zum Beispiel zu einer Rotation den gewünschten Rotationswinkel übergeben, bei der Beschleunigung aber eine Prozentangabe machen wollen.

Als alternative stehen uns die Definition einer gemeinsamen Datenstruktur und die Kontrollkopplung. Die Definition einer gemeinsamen Datenstruktur gestaltete sich im Vorfeld als schwieriges Unterfangen. Kontrollkopplung ist die Übergabe von Datenelementen, die die interne Logik eines Moduls beeinflusst (Raasch, 1991).

(Raasch, 1991) bewertet die Kontrollkopplung als schwach wartbar und schwach wiederbenutzbar, weshalb die Definition einer gemeinsamen Datenstruktur angestrebt werden sollte, im Rahmen dieser Arbeit aber nicht erzielt werden konnte. Da der Inhalt einer Nachricht frei wählbar ist, kann aber zu einem späteren Zeitpunkt eine gemeinsame Datenstruktur erarbeitet werden.

Die Kontrollkopplung hat bereits vom Namen her viel mit dem Begriff Steuerung verwandt, weshalb wir sie im Folgenden mit Vorsicht beim Versand von Nachricht an die Steuerungs-Message-Queue anwenden. Eine Kontrollkopplung schließt Wissen über die Funktionsweise des Empfängers voraus. Die Funktionsweise beschreiben wir in 7.1.3.

Die Kontrolle erfolgt durch die Angabe der Befehle im Label und Optional der Angabe

von Werten im Body. Wird im Body kein Wert angegeben, verwendet das Vehikel sinnvolle Defaultwerte (Beispielsweise rotiere immer um $45°$, bewege um $1$).

Eine einfache Struktur für den Body wären key/value Paare, getrennt per Zeilenende:

```
befehl_als_key  = wert_als_value
befehl2_als_key = wert2_als_value
```

Da Befehle als key dienen sollen, empfiehlt es sich nicht zwei mal den selben Befehl mit unterschiedlichen Werten im Body zu benutzten.

**Beispiele:**

1. Versand einer Nachricht mit einem Bewegungsbefehl:
   Das Vehikel führt die Bewegung mit Defaultwerten aus.
   Beispielcode:

   ```
   send( myobject, forwardZ )
   ```

2. Versand einer Nachricht mit mehreren Bewegungsbefehlen:
   Das Vehikel führt alle Bewegungen mit Defaultwerten in der angegebenen Reihenfolge aus.
   Beispielcode:

   ```
   send( myobject, forwardZ, forwardX, clockwiseX, forwardZ )
   ```

3. Versand einer Nachricht mit mehreren Bewegungsbefehl und Werte im Body:
   Das Vehikel führt alle Bewegungen mit den Bodywerten (wo angegeben, Defaultwert sonst) in der angegebenen Reihenfolge aus.
   Beispielcode:

   ```
   body=new body()
   body.put(forwardZ,0.9)
   body.put(forwardX,0.3)
   send( myobject, forwardZ, forwardX, clockwiseX, forwardZ, body )
   ```

In die andere Richtung, die der Ereignis-Message-Queue, kommen wir um die Definition einer möglichst generischen Datenstruktur nicht herum, da wir nicht genau wissen welche Ereignisse eine Simulation erzeugen kann (Die Simulation ist durch eine beliebig andere Modell austauschbar). Bei der Definition einer Ereignisklasse könne wir uns an bekannte Eventhandling Mechanismen orientieren: Das Ereignis hat eine Bezeichnung, eine Quelle und ein arbiträren Wert.

### 7.1.6 Verwandte Arbeiten

Neben der bereits vorgestellten Entwicklungsframeworks Croquet und Opensimulator, werden hier einige Arbeiten erwähnt, die ähnliche oder verwandte Probleme behandeln.

#### 7.1.6.1 Operator gesteuerte unbemannte Vehikel

(Tso u. a., 1999) entwickelten eine auf einen Multiagensystem basierte Schnittstelle zur Steuerung eines unbemannten Vehikels durch mehrere Operatoren.

#### 7.1.6.2 3D Simulationen

In (Manojlovich u. a., 2003) wurde eine verteilte militärbasierte interaktive 2D Simulation mit einem 3D Computerspiel verbunden, um die Positionierung und Visualisierung von Objekten innerhalb der Simulation in 3D zu ermöglichen.

#### 7.1.6.3 Immersive Datenauswertung

Im *Forum for Telematics & Navigation, Automotive Solutions, Transport & Logistics* der CeBIT 2008 präsentierte (Koester, 2008) Verfahren zur Analyse und Bewertung größerer Datenmengen durch geeignete Speicher- und Visualisierungssysteme. Ein Hauptmerkmal der entwickelten Werkzeuge war die immersive Navigation der Daten zur Auswertung und Kennzeichnung. Weitere Themen waren:

- Viewcars, Fahrzeuge mit auf den Insassen gerichtete Kameras und weitere Sensoren, die zur Studie der Reaktionen in realen Verkehrssituationen dienen.
- Betretbare Verkehrssimulationen, zum Beispiel zur Begutachtung von Verkehrsunfällen durch Verkehrspsychologen.

#### 7.1.6.4 Autonome Systeme

**Car Ring II**

(Richter, 2008) erörtert die Problematik heutiger Intra-Automobil-Kommunikation und stellte ein innovatives System vor, welches folgende Merkmale aufweist:

- Es basiert auf modernen Rechnernetzen, anstatt auf ein Kabelbaum pro Teilsystem.
- Ein Ringbus sorgt für Bandbreite, Ausfallsicherheit und garantierte Auslieferung.
- Das System erfüllt harte Echtzeit, ist nicht TCP/IP basiert
- Hohe Geschwindigkeit. 1GBit im Vergleich zu aktuellen Systemen (1-10Mbit)

- Es Ermöglicht X-by-Wire, beziehungsweise die komplette Fahrzeugsteuerung durch Elektronik
- Niedriges Gewicht . Nur 4 kg Kupfer im Vergleich zu aktuell bis zu 40kg/4km Kabel in einem VW Phaeton.
- Abwärtskompatibel zu den älteren CAN- und FLEX-BUS
- Authentifizierungsmechanismen für Modul2Modul Kommunikation . Dadurch wird verhindert beispielsweise, dass der Sony Lautstärkeregler die Kontrolle über die Bosch Einspritzung übernimmt.
- Implementiert als FPGA , da es als Hardwareguß noch zu teuer ist.

**Spirit Of Berlin**

Das Team der Spirit Of Berlin[34], Semifinalisten der letzten DARPA Grand Urban Challenge, präsentierten ebenfalls auf der CeBIT neben dem beeindruckend ausgestatteten Wagen, auch dessen Simulationumgebung. In dieser Desktopanwendung war die Umgebung ein Foto des Grand Urban Challenge Geländes aus der Vogelperspektive, auf dem ein OpenGL Layer mit dem fahrenden Wagen zu sehen war. Die befahrbare Strecke wurde durch Punkte dargestellt, die neben GPS Informationen auch mit Hilfe eines Regelwerks bestimmten, zu welchen Punkten weiter gefahren werden durfte. Regelwerk und GPS Daten wurden von der DARPA allen Teilnehmern bereitgestellt. Die Richtungsangabe der Punkte war aber nicht immer verlässlich, auf einigen Stellen der Karte war die Punktauflösung so gering, dass die ausgehenden Richtungspfeile bei einer Kurve abseits der Strecke zeigten. Hier spielte Spur-Erkennung die entscheidende Rolle. Realisiert wurde dieser Simulator auf Basis von OpenGL und OpenScenegraph, war aber nur für einen Wagen ausgelegt.

#### 7.1.6.5 FAUST

Die Steuerung fahrerloser autonomer Systeme wird gemäß Subsumption Architektur implementiertPareigis u. a. (2007). Subsumption ist eine Architektur rein reaktiver Agenten: Aus einer gegebenen Situation erfolgt eine Reaktion. Subsumption ist einfach, wirtschaftlich, nachvollziehbar, robust und elegant.

(Wooldridge, 2002) erwähnt aber auch einige fundamentale, unlösbare Probleme, darunter:

- Es ist schwer zu sehen wie rein reaktive Agenten entwickelt werden können, die aus ihren Erfahrungen lernen, und ihre Fähigkeiten im Laufe der Zeit verbessern.

[34] http://robotics.mi.fu-berlin.de/pmwiki/Main/Courses

- Dass sich das Verhalten erst aus lokaler Interaktion mit der Umgebung ergibt, lässt darauf schliessen das die Relation zwischen Agent und Umgebung in Subsumption nur schwer modellierbar ist. Es gibt keine allgemeinere Methodologie beim Entwurf eines reaktiven Agenten, als der Trial and Error Prozess.

## 7.2 Technische Analyse

Im folgenden werden Technologien untersucht, die für die Implementierung der einzelnen Komponenten in Frage kommen. Wir beschränken uns dabei auf die Domäne folgender bevorzugter Entwicklungssprachen:

### 7.2.1 Entwicklungssprachen

#### 7.2.1.1 Java

(frei nach (Böhm, 2002))

- Vorteile
  - Keine nicht-objekt-orientierten und problematischen Features (Makros, Header-Dateien, globale Variablen)
  - Automatisches Memory Management, Garbage Collection
  - Typ-Sicherheit
  - JADEX Planbibliotheken werden (am einfachsten) in Java geschrieben
- Nachteile
  - Ungeeignet für Realtime-Anwendungen
  - Tuning Speicher- und I/O-intensiver Programme nicht möglich.
  - 3D und Dynamics sind überwiegend in C++ implementiert

#### 7.2.1.2 C++

- Vorteile
  - Vielzahl mächtiger Graphics- und Dynamics-Engines
  - Hardwarenahe Optimierung möglich, durch Einsatz von C + ASM code
- Nachteile

  - Eingeschränkte Plattformunabhängigkeit, da es keine VM nutzt; wiederholte Kompilierung
  - Wegen Pointer und fehlendem Garbagecollector fehleranfälliger, nicht nur für Anfänger

#### 7.2.1.3 C#

- Vorteile
  - Interessante Implementierung des *Events and Delegates* Pattern [35]
  - Einige Frameworks wie MS Robot Studio und XNA wären womöglich gute Werkzeueg für die Bearbeitung einiger Aspekte diese Arbeit
- Nachteile
  - Werkzeuge nur für Windows
  - Weitergehende Plattformeinschränkung: Die Laufzeit-VM für C# ist für Microsoft Systeme und nur teilweise für Linux verfügbar. Noch gibt es erhebliche Unterschiede zwischen .NET und Mono.

### 7.2.2 Technologien einzelner Komponenten

#### 7.2.2.1 Modelformate

Objekte in der virtuellen Welt, deren Form komplexer als primitive geometrische Formen wie Quader oder Sphären sind, können mit Hilfe freier Werkzeuge wie Blender[36] erstellt werden. Nicht nur die Form, auch Animationen können im Modell enthalten sein. Im Folgenden wird der Aspekt der Animation aber nicht weiter vertieft.

Die geometrischen Eigenschaften der Modelle können in unterschiedlichen Formaten beschrieben und gespeichert werden, die in ihrer Größe und Komplexität sehr stark variieren. Es ist bei der Wahl einer Rendering-Engine nicht davon auszugehen, das sie alle Modelformate unterstützt. Die zwei populärsten Formate sind:

**OBJ**

Wavefront OBJ ist eines der verbreitetsten Formate und wird von den meisten Modellierungswerkzeugen unterstützt. Objektdateien können in ASCII-Format (.obj) oder Binärformat (.mod) gespeichert werden. Die Spezifikation des ASCII Formats[37] ist offen. Das For-

[35] http://msdn.microsoft.com/en-us/library/17sde2xt(VS.71).aspx Letzter Stand: 19.10.2008

[36] http://www.blender.org

[37] ftp://ftp.oreilly.de/pub/examples/english_examples/gff/CDROM/GFF/VENDSPEC/WAVEOBJ/OBJ_SPEC.TXT

mat stammt ursprünglich von der Firma Wavefront Technologies, die von Silicon Graphics zusammen mit der Firma Alias aufgekauft und zu einer verschmelzt wurde.

#### Das X3D/VRML Format

VRML ist ein Format zur Beschreibung von 3D Szenen. Es wurde ursprünglich als Standard für die Darstellung drei dimensionaler Szenen im Internet entwickelt. Das Dateiformat (.wrl) kann in ASCII oder UTF-8 kodiert sein. X3D bildet den offiziellen Nachfolger des VRML-Standards und ist seit Dezember 2004 als ISO-Standard spezifiziert[38]. Es orientiert sich mehr an der Syntax von XML. Das Format wurde maßgeblich vom Web3D Consortium[39] entwickelt.

### 7.2.2.2 Renderingengines

Mit Rendering sind die Schritte gemeint, die zur Darstellung einer 3D-Computergrafik notwendig sind. Dabei werden Befehle zwischen mehreren Soft- und Hardware Schichten durch gereicht. Die Renderingengine ist eine der Softwareschichten, üblicherweise die letzte zwischen Applikation und Hardwaretreiber. Die im professionellen meist verwendete Programmierschnittstelle ist die weitgehend plattformunabhängige OpenGL Spezifikation.

#### OpenGL

OpenGL ist eine Spezifikation zur Beschreibung von 3D-Computergrafik. Die API wurde auf einer Vielzahl unterschiedlicher Hardware implementiert. Die Firma Silicon Graphics[40] ist maßgeblich an der Spezifikation der OpenGL API beteiligt. Die Referenz-Implementierung der API wurde von Silicon Graphics unter einer Open-Source-Lizenz[41] freigegeben.

#### JOGL

JOGL[42] ist eine direkte Anbindung der OpenGL API an Java. Die Spezifikation wird unter der JSR231[43] noch weiterentwickelt. JOGL erlaubt die Verwendung von 3D Graphik in den AWT und Swing GUI-Bibliotheken.

[38] http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/
[39] http://www.web3d.org/x3d/specifications/x3d_specification.html
[40] http://www.sgi.com/products/software/opengl/
[41] http://oss.sgi.com
[42] https://jogl.dev.java.net/
[43] http://jcp.org/en/jsr/detail?id=231

#### 7.2.2.3 Szenengraphen

**Java3D**

Die Java3D API ist eine Programmierschnittstelle zur Erstellung 3-Dimensionaler Graphiken in Anwendungen und Applets[44]. Java3D wird aktuell als Open-Source Projekt weiterentwickelt[45]. Das Projekt ist in mehrere Module unterteilt. Die drei wichtigsten sind:

1. j3d-core: Kern der Java3D Bibliotheken[46]
2. vecmath: Vektorenmathamitk für Java3D Anwendungen[47]
3. j3d-core-utils: Werzeuge für die Erstellung von Java3D Anwendungen[48]

Als Renderingengine unterstützt j3d-core JSR231/JOGL und Direct3D.

Ein interessanter Aspekt von Java3D ist das Deployment als Java Webstart Anwendung. Man stelle sich folgendes Szenario vor: Während in einem CAVE die Leitzentrale angezeigt wird, meldet ein Entwickler 600 km weiter entfernt einen Agenten zur Steuerung eines Vehikels beim System an. Wie kann der Entwickler an der Visualisierung teilhaben?

Durch Einsatz von j3d-webstart[49] wäre die Bereitstellung eines Clients via Browser möglich, der die Umgebung rund um das Vehikel in 3D visualiseren kann. Der Client müsste zusätzlich zur Visualisierung den Inhalt der virtuellen Welt mit der virtuellen Welt in der Leitzentrale synchronisieren. Die Erstellung eines solchen Clients wäre eine denkbare Erweiterung dieser Arbeit.

**Xith3D**

Xith3D[50] basiert zum größten Teil auf die Java3D API, ist aber zielgerichteter für die Programmierung von Spielen entwickelt worden. Es unterstützt JOGL und LWJGL (Light-Weight-Java-Gaming-Library) als Renderingengines. Xith3D verfügt über praktische Abstraktionsklassen und eigens optimierte Datenstrukturen für Szenengraphoperation. Auch mehrere Beispielimplementierungen sind in der frei beziehbaren Codebasis (BSD Lizenz) vorhanden. Weitere Bibliotheken für Audio, Input und Physics (joode) sind integriert. Auch die nativen Bibliotheken werden in binärer Form im Coderepository mitbetreut. Dahinter steckt die Ansicht, dass die Bibliotheken zwar im einzelnen beziehbar sind, es aber vorkommen kann, dass durch Versionswechsel die Kompatibilität beeinträchtigt wird.

[44] http://java.sun.com/javase/technologies/desktop/java3d/forDevelopers/J3D_1_3_API/j3dguide/Introduction.html
[45] https://java3d.dev.java.net/
[46] https://j3d-core.dev.java.net/
[47] https://vecmath.dev.java.net/
[48] https://j3d-core-utils.dev.java.net/
[49] https://j3d-webstart.dev.java.net/
[50] www.xith3d.org

Die Betreuung nativer Bibliotheken ist ein hervorzuhebender Aspekt, denn die Handhabung jeder einzelnen Bibliothek kostet Zeit. Zu den Nachteilen gehören die erwähnte Ausrichtung auf Spiele und kein direkter Support für Multiscreen Darstellung (CAVE).

Es folgt ein Auszug der für uns Interessanten, im Xith3D Repository mitbetreuten Bibliotheken:

```
 |-- gluegen
|   |-- gluegen-rt.jar
|   |-- linux-i586
|   |   `-- libgluegen-rt.so
|   `-- windows-i586
|       `-- gluegen-rt.dll
|-- input
|   `-- jinput
|       `-- jinput.jar
|-- jagatoo.jar
|-- jogl
|   |-- jogl.jar
|   |-- linux-i586
|   |   |-- libjogl.so
|   |   |-- libjogl_awt.so
|   |   |-- libjogl_cg.so
|   |   `-- libjogl_drihack.so
|   `-- windows-i586
|       |-- jogl.dll
|       |-- jogl_awt.dll
|       `-- jogl_cg.dll
|-- math
|   `-- openmali.jar
|-- particles
|   `-- jops.jar
`-- physics
    `-- joode.jar
```

#### 7.2.2.4 Simulation

**ODE**

Die *Open-Dynamics-Engine* ist eine Opensource Bibliothek zur Simulierung von Körperdynamik[51]. Mit Hilfe dieser Bibliothek können Kollisionen und weitere dynamische Effekte in virtuellen Umgebungen simuliert werden.

**JOODE**

Die JOODE Bibliothek ist eine Portierung der ODE Bibliothek in Java[52].

#### 7.2.2.5 Multiagentsysteme

**Jadex**

Jadex ist eine rückkompatible Erweiterung des Java Agent DEvelopment Framework (JADE)[53], entwickelt an der Universität Hamburg[54]. Die Middleware erlaubt die Entwicklung und Ausführung von MAS in Java.

#### 7.2.2.6 Roboter

**Robotino**

Robotino[55]. ist ein Roboterkit für den Bildungsbereich.

#### 7.2.2.7 Kommunikation

**CORBA**

Die *Common Object Request Broker Architecture*[56] ist eine Technologie der Object Management Group, Inc.[57]. Es erlaubt unter anderem den Aufruf von Methoden entfernter Objekte über Sprachgrenzen hinaus. Schnittstellen werden mit Hilfe einer eigenen Interface-Definition-Language (IDL) spezifiziert. Beim RPC über CORBA wird angenommen, dass neben Object-Request-Broker und Naming-Service auch der entfernte Prozess aktiv sein muss.

[51] http://ode.org/
[52] http://joode.sourceforge.net/index.html
[53] http://jade.tilab.com/
[54] http://vsis-www.informatik.uni-hamburg.de/projects/jadex/
[55] http://www.festo-didactic.com/de-de/lernsysteme/neu-robotino-lernen-mit-robotern/
[56] http://www.corba.org/
[57] http://www.omg.org/

Nach (Tanenbaum und van Steen, 2008) bot CORBA anfangs nur Objekte, bis man merkte, dass es zu einschränkend war und fügte Nachrichten hinzu. Daraus wurde offensichtlich, wie

> das Hinzufügen neuer Eigenschaften leicht zu aufgeblasenen Middleware-Lösungen führen

kann, ein Grund weshalb auf den Einsatz von CORBA verzichtet wurde.

#### XML-RPC

Das XML-RPC[58] wurde zur Kommunikation über das Internet in Form von HTTP-POSTs entwickelt. Es setzt auf beiden Seiten neben den Einsatz von Webservern XML Parser voraus.

#### JMS

Das Java-Messaging-System [59] (kurz JMS) entstand aus der Forderung (JSR914[60]) nach einer einheitlichen Schnittstelle zu nachrichtenorientierten Middlewaresystemen. Es erlaubt die Nutzung zweier Modelle zum Austausch von Nachrichten, Point-to-Point (Mailbox) und Publish/Subscribe. Einige Implementierungen des JMS sind frei verfügbar und kostenlos nutzbar. Kostenpflichtige Enterprise Varianten versprechen bessere Skalierbarkeit [61] und höhere Ausfallsicherheit.

#### Sun Java System Message Queue

Ein Problem mit Suns JMS Implementation[62] ist die ausschließliche Unterstützung von Java Clients. Es gibt zwar in C++ nutzbare Clients wie Junction[63], die aber weiterhin eine JVM (Java-Virtual-Machine) benötigen. Insbesondere Roboter mit embedded Controller besitzen oft keine JVM, da nicht alle JVM Implementationen echtzeitfähig sind (Böhm, 2002). Eine schlankere Lösung, die keine JVM auf Seiten des Clients voraussetzt wäre wünschenswert.

[58] http://www.xmlrpc.com/spec

[59] http://java.sun.com/developer/technicalArticles/Ecommerce/jms/index.html

[60] http://www.jcp.org/en/jsr/detail?id=914

[61] Das Versprechen der Skalierbarkeit ist aufgrund der Auslegung als Client-Server Anwendung nur bedingt glaubwürdig (Tanenbaum und van Steen, 2008)

[62] http://www.sun.com/software/products/message_queue/index.xml

[63] http://www.codemesh.com/products/junction/examples/jms.html

### Apache ActiveMQ

Die Apache ActiveMQ[64] beherrscht neben JMS eine Vielzahl weiterer Protokolle, durch die Clients in den meisten Programmiersprachen angebunden werden können. Die Serveranwendung ist in Java implementiert.

### SAFMQ

Die *Store And Forward Message Queue* [65] ist eine in C++ geschriebene Message Queue ohne Broker. Das Programm wird zu einem Server, einem Client und einer Bibliothek kompiliert. Externe Abhängigkeiten besitzt es nach eigenen Angaben nur zur OpenSSL Bibliothek. Diese Abhängigkeit lässt sich durch Angabe entsprechender Konfigurationsparameter beim Kompilieren abschalten. Die Anbindung an eine SAFMQ Server erfolgt über den nativ kompilierten Client[66] oder über die C++, Java, PHP und C# Schnittstellen. Der Autor des Opensource Projekts gibt an, derzeit mit der Implementierung des JMS beschäftigt zu sein.

SAFMQ bietet ein interessantes Konzept für Round-Trip, PseudoSynchronous Messaging[67]: beim Einreihen der Nachrichten in die Message-Queue bekommt diese vom Server eine UUID, die der Sender als RecipientID zum Abholen einer Antwort auf einer spezifischen Nachricht verwenden kann. Dazu muss der Sender die Adresse einer Antwort-Queue nennen.

Der Sender kann die Lebensdauer (TTL) einer Nachricht bestimmen. Die Zeit wird ab Eintritt auf dem Server gemessen. Verarbeitet der Empfänger bis zum Ablauf der TTL die Nachricht nicht, bekommt der Sender eine Fehlermeldung vom MessageQueue Server in einer Antwort-Queue.

### MPI

Das Message-Passing-Interface[68] ist eine Spezifikation für Prozesskommunikation im Bereich hocheffizienter paralleler Programmierung. Zwar ist Interprozesskommunikation prinzipiell möglich, entwickelt wurde MPI aber für die Koppelung von Hardwaresystemen, die über MPI verteilt an der selben Aufgabe rechnen (Sloan, 2005).

[64] http://activemq.apache.org/
[65] http://safmq.sourceforge.net
[66] Der Client wird direkt in die Maschinensprache des Zielsystems übersetzt
[67] http://69.10.233.10/KB/IP/safmq_roundtrip.aspx
[68] http://www-unix.mcs.anl.gov/mpi/

### 7.2.3 Entwicklungsframeworks / SDKs

#### 7.2.3.1 Croquet

**Beschreibung**

Croquet [69] ist ein Open-Source Entwicklungs-Kit zur Erstellung verteilter kollaborativer virtueller Welten. Als Derivat von Squeak ist die Entwicklungssprache Smalltalk. Croquet verwendet das Konzept von *Islands*, eine Art Container für Objekte. Für den Betrachter wird der Inhalt einer Insel in 3D lokal gerendert. Jede Insel verfügt über eine Message-Queue, durch die interne und externe Kommunikation abgehandelt wird. Eine besondere Eigenschaft der Islands ist deren Replizierbarkeit. So lassen sich alle Objekte, deren Zustand und Nachrichten in der Queue auf Festplatte speichern oder übers Netzwerk replizieren. Croquet ist in der Lage, komplexe Simulationen und Interaktionen auf alle Repliken einer Insel in diskreter Zeit auszuführen.

**Fazit**

Das aktuelle Release der Croquet SDK benutzt ein Smalltalk spezifisches Binärformat zur Szenenbeschreibung, weshalb Definition und Interaktion nur aus Smalltalk Clients heraus möglich ist. Die noch nicht abgeschlossene Entwicklung der

> XML Specification for Replicated Environments (XML Scene Description) [70]

soll Croquet Zugriff auf die Islands, auch Clients in anderen Sprachen ermöglichen.

#### 7.2.3.2 OpenSimulator

**Beschreibung**

OpenSimulator ist eine freie Implementierung der Secondlife-Serveranwendung. Damit können kollaborative virtuelle Welten, hier *Regions* genannt, betrieben werden. Mehrere Regions können auf *Grid* Server vereint werden. Ein Beispiel hierfür ist der öffentliche *Osgrid* [71] Die Serveranwendung ist in C# geschrieben und verwendet *Mono*, die freie Implementation des .NET Frameworks für Linux. Die Anbindung an den Grid erfolgt über den Secondlife-Client, dessen Quellcode die Firma *Linden Labs* [72] freigegeben hat. Externe Client lassen sich über das XML-RPC Protokoll verbinden. Der Anwender steigt mit einem Avatar direkt in die virtuelle Welt ein, wo er das Aussehen des Avatars und das der Umwelt bestimmen kann. Ebenfalls ist es möglich, neue Objekte, sogenannte (*Prims*) zu erzeugen und per Scripts zu

[69] http://www.opencroquet.org/index.php/Main_Page Stand 14.10.2008
[70] http://www.opencroquet.org/index.php/Cobalt_Documentation am 14.10.2008
[71] http://osgrid.org
[72] http://secondlifegrid.net/programs/open_source/faq

steuern oder zu animieren. Das erlaubt dem Anwender sich an der Gestaltung der virtuellen Welt zu beteiligen. Serverseitig bietet OpenSim eine Pluginarchitektur, die die Einbindung eigener Module beziehungsweise *RegionModules* ermöglicht. Zur Erstellung eines Regionmodules gehört die Implementierung des IRegionModule Interfaces und die Registrierung für bestimmte Ereignisse über das C# Event/Delegate Pattern. Siehe dazu folgenden Democode:

```
public interface IRegionModule
{
    void Initialise(Scene scene, IConfig config);
    void PostInitialise();
    void Close();
    string Name { get; }
    bool IsSharedModule { get; }
}
...
public class TestModule : IRegionModule
{
...
  // Interface Methode implementieren
  public void Initialise(Scene scene, IConfigSource config)
  {
    ...
    // Event / Delegator Pattern in C#:
    // Auf das Ereignis OnFrame
    // wird unsere Methode TestUpdate augerufen
    scene.EventManager.OnFrame      += TestUpdate;
    ...
  }
...
  public void TestUpdate()
  {
     // Mach etwas in der Region
     // Sollte schnell fertig sein, hier ist Frametime (Renderloop)!
  }
...
}
```

**Fazit**

Zu der Frage in der Entwickler-Maillist, ob OpenSim für physikalisch korrekte Simulatoren verwendet werden kann [73], wurde von einem der Entwickler kommentiert, dass die Einbindung der Physicsengine in OpenSim generell nicht für physikalisch korrekte Simulationen ausgelegt sei. OpenSim könnte eher als Schnittstelle zu Systemen fungieren, die komplexe Simulationen ausführen und große Datenmengen zur Auswertung hinterlassen. Eine Teilnehmerin an der Diskussion erwähnte ein Projekt zur Verkehrsimulation[74] in OpenSim realisiert zu haben, in dem Sie letzendlich den Vehikeln Phantomkörper (durchsichtige, unbeteiligte Simulationskörper) verpassen musste, weil die Physicsengine in OpenSim zu hart auf die Körper wirkte.

OpenSim macht einen sehr guten Eindruck, befindet sich aber noch sehr stark in der Entwicklung und der Quellcode ist so gut wie gar nicht dokumentiert. Es gibt eine Reihe von TestSuites, die den Einstieg anhand von Democode erleichtern.

So wurde eine TestRegion erstellt, in der per Zufall gesteuerte Avatare herumliefen. Mehr als Avatare liess sich aber auch nicht steuern. Das Konzept der Vehikel lässt sich so noch nicht implementieren. Die Prims können zwar geskriptet werden, die Möglichkeiten der eingebauten Skriptengine sind aber recht limitiert.

## 7.3 Zusammenfassung

In der fachlichen Analyse wurde nachrichtenorientierte Kommunikation näher betrachtet. Es wurde festgestellt, dass sich für den zeitlich entkoppelten Betrieb Message-Queues eignen, in denen Nachrichten bis zu ihrer Abholung in der Reihenfolge der Ankunft gespeichert werden. Auf referenzielle Entkoppelung durch das Publish/Subscribe Modell wurde verzichtet. Statt dessen soll das Mailbox Modell verwendet werden, so dass jeder Empfänger eine eigene Message-Queue bestimmen oder sich diese mit anderen Empfängern teilen kann. Die Steuerungsnachrichten an Vehikeln wurden in Form einer Kontrollstruktur spezifiziert. Für Ereignismeldungen wird eine möglichst generische Datenstruktur verwendet. Das Kapitel wurde mit einem Überblick verwandter Arbeiten abgeschlossen.

Die Entwicklungsframeworks OpenSim und Croquet erfüllen einige kritische Anforderungen nicht. OpenSim kann über das XML-RPC heterogene System verbinden. Die Nachrichten sind aber zunächst nicht persistent, eine Datenbank-Anbindung oder ähnliches wäre nötig. Auch der TTL Parameter müsste zusätzlich implementiert werden. Die virtuelle Welt in OpenSim bietet gute Interaktionsmöglichkeiten und immersive Gestaltung. Die eingebaute Skriptengine für die Kontrolle der Prims ist aber nicht so mächtig in Vergleich zu Smalltalk in Croquet oder die direkte Kontrolle über einen Szenengraphen. Croquet bietet persistente

[73] https://lists.berlios.de/pipermail/opensim-dev/2008-September/002848.html

[74] http://www.ics.uci.edu/ lopes/images/traffic-control/

Erfüllung der Anforderungen durch Frameworks

| **Visualisierung** | **Anforderung** | **Croquet** | **Opensim** |
|---|---|---|---|
| muss | OpenGL verwenden | x | x |
| soll | Interaktionshandling bieten | x | x |
| soll | Einfache Anbindung Jadex (Java) | | |
| soll | Mehrere Kameraperspektiven unterstützen | x | x |
| kann | Verteilt Rendern | | |

| **Simulation** | **Anforderung** | | |
|---|---|---|---|
| muss | unabhängig vom System | x | x |
| soll | Modell austauschbar | x | x |
| soll | Objekt nimmt Teil oder nicht | x | x |
| kann | Kollisionen berechnen | x | x |

| **Kommunikation** | **Anforderung** | | |
|---|---|---|---|
| muss | Heterogene Systeme verbinden | | x |
| muss | Agent unabhängig | | x |
| soll | Lose Koppelung (zzgl. asynchron+persistent) | x | |
| soll | Diskrete Zeit (weiche Echtzeit) | x | |
| soll | TCP/IP basiert sein | x | x |

X = Erfüllt Anforderung

Tabelle 7.1: Gegenüberstellung Technologie/Anforderungen

Nachrichten und Auslieferung in diskreter Zeit, leidet aber aus Sicht heterogener System an der starken Smalltalkbindung.

Die Erfüllung der technischer Anforderungen 6.2 durch die einzelnen Technologien wird in den Tabellen 7.1 und 7.2 nochmal zusammengefasst dargestellt.

Erfüllung der Anforderungen durch die einzelnen Technologien

| Visualisierung | Anforderung | OpenGL | JOOGL | Java3D | Xith3D |
|---|---|---|---|---|---|
| muss | OpenGL verwenden | x | x | x | x |
| soll | Interaktionshandling bieten | | | x | x |
| soll | Einfache Anbindung Jadex (Java) | | | x | x |
| soll | Mehrere Kameraperspektiven unterstützen | x | x | x | x |
| kann | Verteilt Rendern | * | | | |

| Simulation | Anforderung | ODE | JOODE |
|---|---|---|---|
| muss | unabhängig vom System | x | x |
| soll | Modell austauschbar | x | x |
| soll | Objekt nimmt Teil oder nicht | x | x |
| kann | Kollisionen berechnen | x | x |

| Kommunikation | Anforderung | CORBA | XMLRPC | SAFMQ | SUNsMQ | ActiveMQ |
|---|---|---|---|---|---|---|
| muss | Heterogene Systeme verbinden | x | x | x | | x |
| muss | Agent unabhängig | x | x | x | x | x |
| soll | Lose Koppelung (zzgl. asynchron+persistent) | | | x | x | x |
| soll | Diskrete Zeit (weiche Echtzeit) | | | x | x | x |
| soll | TCP/IP basiert sein | x | x | x | x | x |

X = Erfüllt Anforderung
*) Bestimmte OpenGL Implementierungen

Tabelle 7.2: Gegenüberstellung Technologie/Anforderungen

# Kapitel 8

# Architekur

## 8.1 Architekturstil

Architekturen bieten wiederverwendbare Baupläne bei der Entwicklung von Software. Sie beschreiben Lösungen in einer verständlichen Form.

Wir verfolgen die Idee, zeitlich und räumlich entkoppelte Systeme, Nachrichten und Ereignisse austauschen zu lassen, weshalb sich unsere Architektur zunächst dem Stil ereignisbasierter Architekturen einorden lässt (Tanenbaum und van Steen, 2008).

Diese ereignisbasierte Architektur auf der Ebene der VR und Vehikel, baut durch den Einsatz von Message-Queues auf ein nachrichtenorientierte Middlewaresystem, welche als Client-Server Architektur realisiert ist. Die Message-Queue ist der Server, Clients sind Sender und Empfänger von Nachrichten.

Die VR Komponente lässt sich als MVC Architektur implementieren: Die Simulation als *Modell*, die Visualisierung als *View* und die Nachrichtenverarbeitung und steuernde GUI Elemente als *Controller*.

Die Architektur des entwickelten Systems zeigt Abbildung 8.1.

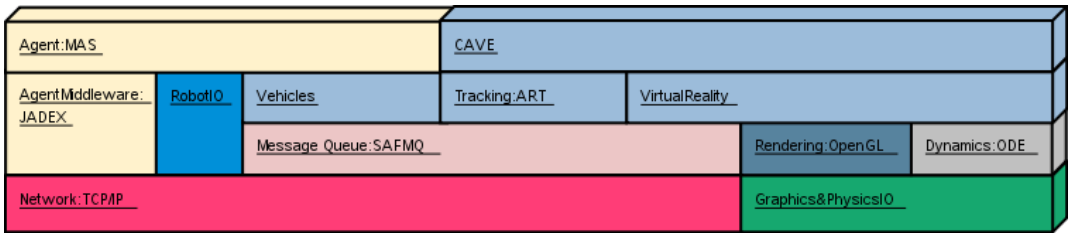


Abbildung 8.1: Architektur

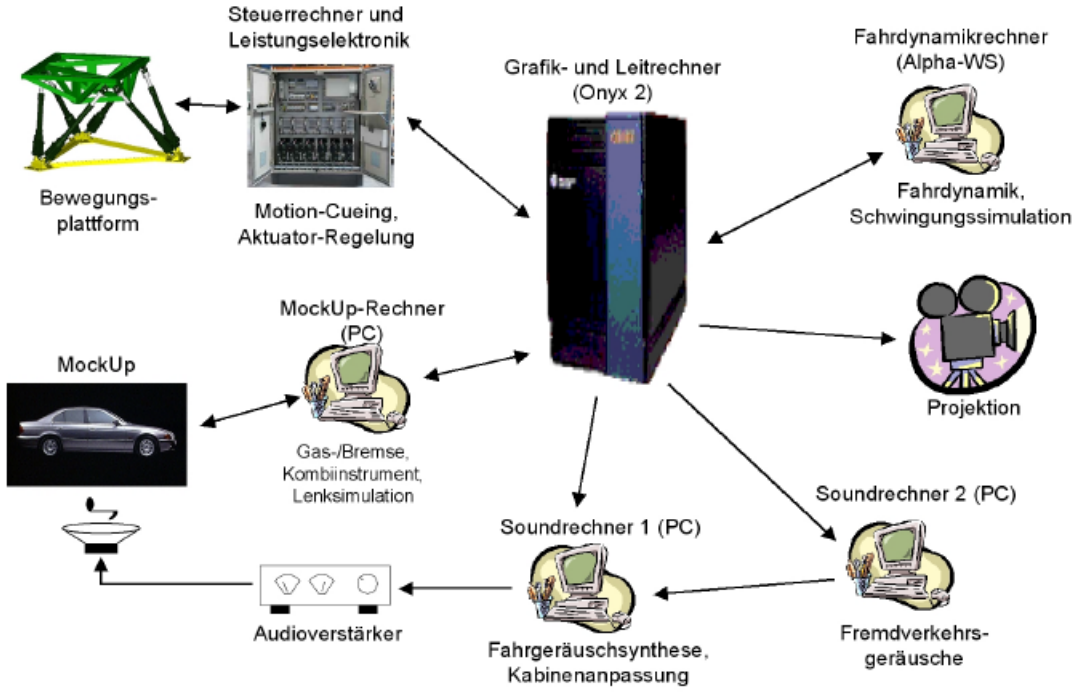


Abbildung 8.2: Struktureller Aufbau des BMW Schwingungs- und Akustiksimulator

## 8.2 Vergleichbare Architekturen

Der Blick über den Tellerrand auf andere, bestehende Architekturen hilft uns, Entscheidungen für das eigenen System zu überdenken, aus Fehlern zu lernen und Best-Practices anzuwenden.

### 8.2.1 Die Architektur des BMW Schwingungs- und Akustiksimulators

stellen Kirchknopf und Witta in (Kirchknopf und Witta, 2003) vor (Abbildung 8.2). Die saubere Trennung der Verwantwortlichkeiten der Teilsysteme ist auf den ersten Blick zu erkennen. Leider wird nicht erwähnt, wie die Kommunikation realisiert wird, nur dass alle Komponenten mit dem Leitrechner über *Netzwerk* verbunden sind. Es lässt sich erkennen dass die Erweiterbarkeit und Skalierbarkeit des Systems sehr stark vom Leitrechner abhängt. Ein Problem auf dem Leitrechner wirkt sich auf das gesamte System aus.

# Kapitel 9

# Realisierung

Die Machbarkeit einer über Message-Queues erreichbaren VR Umgebung konnte in einem Prototyp demonstriert werden.

Wegen der Anforderung die Engine möglichst als ein Umgebungsprogramm für Jadex Agenten zu realisieren, schränkte sich die Auswahl für die Visualisierungslösung auf Java3D und Xith3D ein. Erste Tests mit der Java3D Applikation Wonderlandhttps://lg3d-wonderland.dev.java.net/ verliefen viel versprechend, doch die OpenGL Anbindung war geprägt von Treiberabstürzen[75][76].

Xith3D erlaubte es anders als Java3D neben dem Szenengraphen auch direkten Einfluss auf OpenGL Parameter zu nehmen. So zeigte sich, dass durch die Deaktivierung von DisplayLists JOGL stabil lief. Auch wegen der restlichen mitbetreuten nativen Bibliotheken und letztlich auch aus Zeitgründen stellte Xith3D die bessere Wahl. Allerdings eignet sich die Engine nicht für CAVEs oder Multiscreendarstellung. Dafür können virtuelle Umgebungen Plattformunabhänig auf dem *lokalen* Bildschirm erstellt werden.

Die Kommunikation wurde über SAFMQ erledigt. Es wurd auf die größeren Technologien verzichtet da diese in der Prototypphase kein echten Mehrwert brachten, sondern im Gegenteil, eine Menge Overhead. ActiveMQ könnte sich aber wegen des vielseitigen Brokers sicherlich besser für künftige Anforderungen eignen.

Die Entwicklung des Prototyps erfolgte *evolutionär* (GILB-88 in (Raasch, 1991, S. 415)): während der Bearbeitung der Arbeit wurden einzelne Lösungen der Teilprobleme (Kapitel 2.4) implementiert. Die Ausführung aller Komponenten ist möglich, ohne aber alle in dieser Arbeit entwickelten Ziele auf einen Schlag zu erfüllen. So blieb keine Zeit übrig, die Simulationskomponente befriedigend zu integrieren.

Das Prototyp dient primär dazu die Machbarkeit des System zu demonstrieren, Peformancemessungen durchzuführen und als Basis für eine Weiterentwicklung dienen.

[75] https://java3d.dev.java.net/issues/show_bug.cgi?id=565

[76] Treiberprobleme gibt es auch unter Linux, immer dann wenn Unternehmen geschlossene proprietäre Treiber für ihr Produkte heraus geben, auf die aber Leider nicht immer verzichtet werden kann.

## 9.1 Bewertung

### 9.1.1 Sytemintegration

Das entwickelte System ist in der Lage eine rudimentäre VR Umgebung auf dem Computerbildschirm zu präsentieren. Die Anwendung kann auch direkt von Jadex als Umgebungsprogramm für Agenten verwendet werden, da alle Komponente außer der Message-Queue in Java implementiert wurden. Beim lokalen Aufruf der Visualisierungsimplementation wird die Message-Queue nicht benötigt.

Durch die Java-Implementation ist es möglich die VR Umgebung aus einem Guß auf Linux und Windows auszuführen. Bibliotheken werden auch für Mac und Solaris mitgeliefert, die aber nicht getestet werden konnten.

Die Installation von SAFMQ ist auf den meisten Systemen problemlos möglich. Der Server braucht im Prinzip nur einmal in der Infrastruktur vorzukommen, idealerweise auf der selben Platform, in der die VR Umgebung ausgeführt wird. Der Versand von Nachrichten über einem nativ übersetzen client dürfte für die meisten Plattformen, die mindestens über TCP/IP Konnektivität verfügen, kein Problem darstellen. Es stehen Alternativ ein C++, Java, PHP und C# Interface zur Verfügung.

Das entwickelte System erlaubt es einer Vielzahl unterschiedlicher Plattformen, durch den Versand persistenter Nachrichten, eine Präsenz in die virtuelle Simulation zu bringen. Die dafür benötigte Operation aus Seiten des Clients kosten wenig Resourcen und haben außer dem SAFMQ-Betrieb keine weiteren Abhänigkeiten.

Das System ist unabhänig von der präferieten Agentarchitektur; ob Subsumption oder BDI:
Das vorgestellte System ist vom Aufbau sehr unabhängig, weshalb problemlos beide Architekturen zur Steuerung von Vehikel und Ereignisauswertung virtueller Situationen verwendet werden können. Auch Szenarien in denen beide Architekturen zur Lösung einer Aufgabe gegeneinander antreten sind vorstellbar.

### 9.1.2 Performance

Die erwähnt benötigen Operation aus Seiten des Clients nur wenig Resourcen, Das ist anders für die VR. Da hier die Nachrichten aller Teilnehmer zusammenkommen, ist zu erwarten, dass das Systems in seinere jetzigen Konstellation, ab einer bestimmter Anzahl an Teilnehmern überfordert ist. Die Anzahl konnte in den folgenden Performancemessungen ermitelt werden.

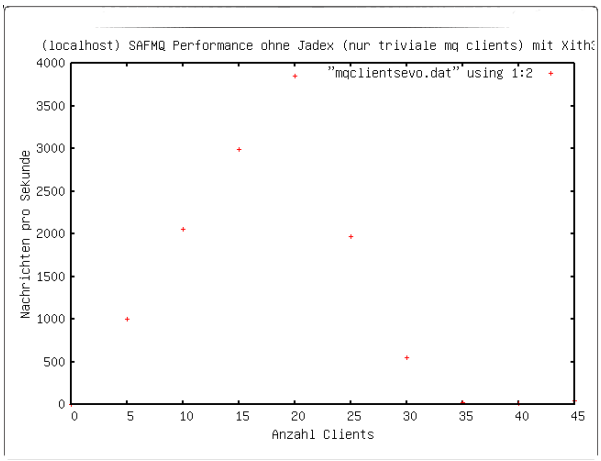


Abbildung 9.1: Nachrichtendurchsatz bei Aufruf aus Java Clients ohne Jadex

### 9.1.3 Erste Testreihe: Nachrichten-Durchsatz

#### 9.1.3.1 Aufbau

JADEX und VR wurden zusammen auf einer Maschine mit Core2-duo 2Ghz Prozessor, 2GB Ram und ATI Mobility Radeon X1700 Grafikbeschleuniger ausgeführt. Der SAFMQ server wurde auf einem Pentium 4 3Ghz System mit SATA-150 Platte betrieben. Beide Systeme waren über ein 100Mbit Ethernet-LAN verbunden.

#### 9.1.3.2 Vorgang

Die VR Umgebung wir gestartet. Etwas alle 30 Sekunden wird ein Agent manuell gestartet. Jeder Agent versucht alle 33 ms den "move"Plan auszuführen, der eine zufällige Bewegungsnachricht an die Queue schickt.

#### 9.1.3.3 Ablauf

Erkennbar in Abbildung 9.1.3.3 ist ein deutlicher Einbruch des Nachrichtendurchsatzes bei steigender Agentenanzahl. Abbildung 9.1.3.3 zeigt, dass der höchste Durchsatz bearbeiteter Nachrichten bei 5 teilnehmenden Agenten lag. Auffallend höher ist der Nachrichtendurchsatz bei direktem Aufruf des Java MessagingClients ohne Jadexagenten, was in Abbildung 9.1.3.3 zu sehen.

#### 9.1.3.4 Auswertung

Im ersten Testdurchlauf wurden Probleme unter anderem bei der Ausführung der Bewegungen festgestellt, was von Sprüngen in der Animation gekennzeichnet war. Zur Eingrenzung

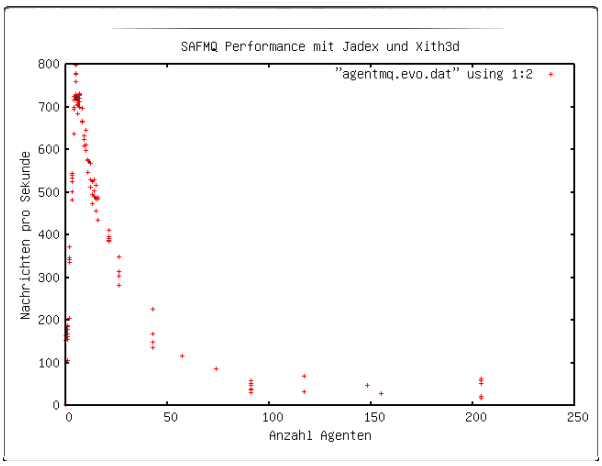


Abbildung 9.2: Verlauf Nachrichtendurchsatz bei wachsender Agentenzahl

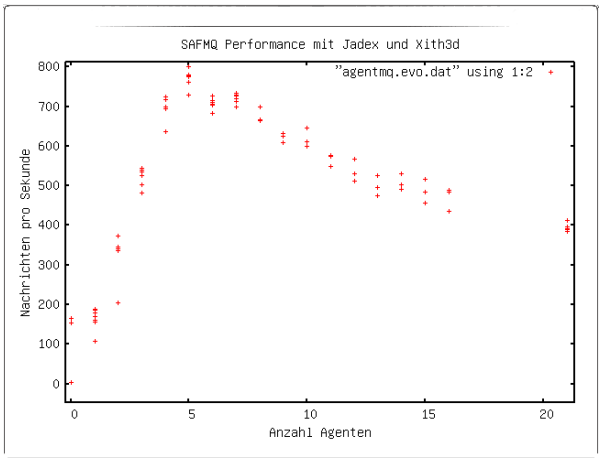


Abbildung 9.3: Nachrichtendurchsatz bei 5 Agenten am höchsten

des Problems wurden die Fehlercodes des Nachrichtenversands genauer ausgewertet. Es stellte sich heraus, dass einzelne Nachrichten nicht versendet wurden. Der Grund hierfür war die Wiederverwendung der geöffneten MessageQueue Facade. Sporadisch trat beim Versand der Nachricht ein Fehler unbekannter Fehler-Code auf. Die Umstellung des Codes auf Öffnen und Schließen der MessageQueue Facade bei jedem Nachrichtenversand schaffte Abhilfe nur bis zu einem bestimmten Punkt, an dem das System komplett stehen blieb. Ein Blick auf 'netstat -tap' verriet den Grund. Per Default geöffnete Java Sockets wechseln beim Schliessen in den Zustand TIMEOUT_WAIT und bleiben so noch ein paar Minuten aktiv. Das System weigerte sich schlicht, weitere Sockets aufzumachen. Aus diesem Grund verwerfen wir die Ergebnisse der ersten Testreihe.

### 9.1.4 Zweite Testreihe: Nachrichtendurchsatz

Die zweite Testreihe sollte eine bessere Auswertung als die Erste Reihe ermöglichen. Dazu wurde Grundlegendes geändert: Die X Achse stellt nicht mehr die Anzahl der Vehikel sondern die Zeit dar. Nachrichtendurchsatz und Vehikelzahl (producer) werden nun zu genauen Zeitpunkten gemessen. Auch wurden keine Agenten mehr eingesetzt, sondern zeitlich exakt steuerbare Vehikel. Ob die Steuerung ein Agent übernimmt betrachten wir als irrelevant. Jeder Test wurde zweimal ausgeführt.

Das Socket Problem wurde gelöst, in dem statt der safmq.MessageQueue Facade die MConnection direkt verwaltet wird. Durch die Kontrolle des Sockets konnte die Wiederverwendung, auch als *Socket-Hijacking* bekannt, festgelegt werden:

```
Socket socket = null;
socket = new Socket();
socket.setReuseAddress( true );
socket.setTcpNoDelay( true );
socket.setPerformancePreferences( 2, 1, 0 );
socket.connect(new InetSocketAddress( uri.getHost(), uri.getPort() ) );
```

Die Animation wurde durch Synchronisierung der Transformation stabilisiert. Dies erlaubt eine weichere und vor allem fehlerfreie Bewegung, kostet aber voraussichtlich Performance.

#### 9.1.4.1 Aufbau

1. Server Maschine: Core2-duo 2Ghz Prozessor, 2GB RAM, Ati Mobility Radeon X1700 Grafikbeschleuniger, eSATA 7200 U/min Festplatte
2. Client Maschine: Athlon XP 3000+ mit 1 GB RAM
3. Netzwerk: 100Mbit Ethernet

#### 9.1.4.2 Vorgang

Eine VR Umgebung wurde gestartet. Alle 60 Sekunden wurden 2 zusätzliche Vehikelthreads gestartet. Nach 10 Durchläufen werden die Vehikelthreads gestoppt. Gemessen wurde bis alle in der Queue befindlichen Nachrichten verarbeitet wurden. 1 Sekunde wurde als weiche Echtzeitanforderung gewählt, weil es der kleinste TTL Wert ist den SAFMQ annimmt. Folgende Variationen wurden durchgeführt:

- t2 und t3
  - Gleichzeitige Ausführung
  - VR, SAFMQ und Vehikel auf dem Server
- t4 und t5
  - Gleichzeitige, verteilte Ausführung
  - SAFMQ und VR auf dem Server
  - Vehikel auf dem Client
- t6
  - Asynchrone, verteilte Ausführung
  - Vehikel auf dem Client
  - SAFMQ und VR auf dem Server (nach dem die Clients alle Nachrichten versendet haben)
  - Entspricht der passiven Simulation

#### 9.1.4.3 Ablauf

Tests t2 und t3 verliefen bis zu einer Grenze von etwa 12 Vehikel gut, die Bearbeitungszeit stieg dann aber ab $t = 30$ exponentiell von unter 1 Sekunde auf bis zu 90 Sekunden. Die Animation war noch sichtbar, aber zwischen Anmeldung eines Vehikels und das Auftauchen in der Simulation vergingen mehrere Sekunden. Die Situation besserte sich erst als die Clients aufgehört haben Nachrichten zu schicken. Eine Ursache sind die konkurrierenden Prozesse auf der lokalen Maschine, wie die verteilten tests t4 und t5 zeigen. Allerdings auch hier ab 12 Vehikel kaum Bearbeitung unter einer Sekunde möglich.

Den Überblick zeigt Abbildung 9.4). In Abbildung 9.5 ist der Zeitpunkt des Sprungs eingegrenzt. Es ist zur erkennen das t4 und t5 noch sporadisch unter 1 Sekunde blieben. Abbildung 9.6 zeigt den Nachrichtendurchsatz während des gesamten Vorgangs. Zum Vergleich ist hier auch zu sehene wie die Leistung ist wenn nur der Server auf die Message-Queue zugreift (t6). Abbildung 9.7 zeigt abschließend, wie viel Zeit zwischen dem Eingang der letzten Nachricht $(t = 60)$ und dessen Bearbeitung vergangen ist.

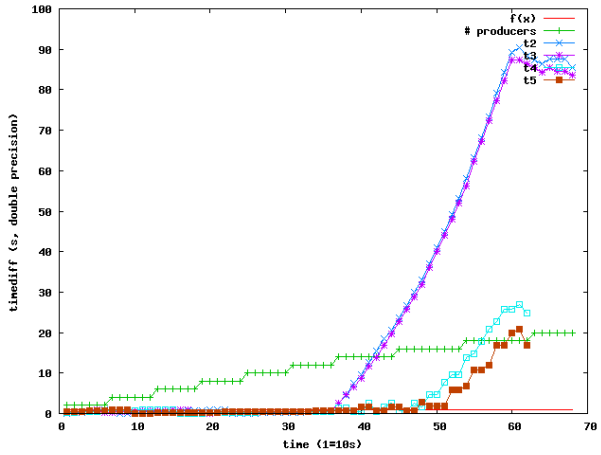


Abbildung 9.4: Zeitdifferenz steigt ab einem *Sätigungspunkt* des Bearbeitungsthreads in t2 und t3 exponentiell, t4 und t5 steigen nicht so extrem, aber auch deutlich über 1 Sekunde.

#### 9.1.4.4 Auswertung

Der spontane Wachstum der Zeitdifferenz zwischen Ankunft und Bearbeitung der Nachricht bedeutet vor allem dass der SAFMQ Server vor dem Überlauf geschützt werden sollte, da es für die Anwendung nicht möglich ist alle Nachrichten abzuarbeiten. Genügend Speicherplatz und Blockade des externen Netzwerkverkehrs auf den SAFMQ port ab einer besimmten Schwelle wäre erste effektive Lösungen.

Die Trennung von Server und Client auf unterschiedliche Maschinen hat die Performance zwar verbessert, ab einer ähnlichen Zahl von Clients konnte das System die Bearbeitung unter 1 Sekunde auch nicht mehr realisieren.

Das System eignet sich in dieser Konstellation, zum Betrieb einer aktiven Simulation zu einer maximalen Teilnehmernzahl von $X <= 10$.

## 9.2 Systemablauf

Erste Voraussetzung für die Ausführung des Systems ist ein aktiver SAFMQ Server und die JRE 6[77]. Installationspakete für Windows, Source-Code und eine auf SuSE Linux kompilierte Version sind im angehängten Datenträger dieser Arbeit enthalten. Ein aktuelle Version von SAFMQ kann auch von der Projekt Homepage[78] bezogen werden.

Die Kompilierung auf einem Debian 4 (lenny) erfolgt ohne Probleme, wenn die STL Libraries installiert sind. Auf OpenSuSE 11 werden anfangs nicht alle benötigten

[77] http://java.sun.com/javase/6/webnotes/install/index.html
[78] http://safmq.sourceforge.net

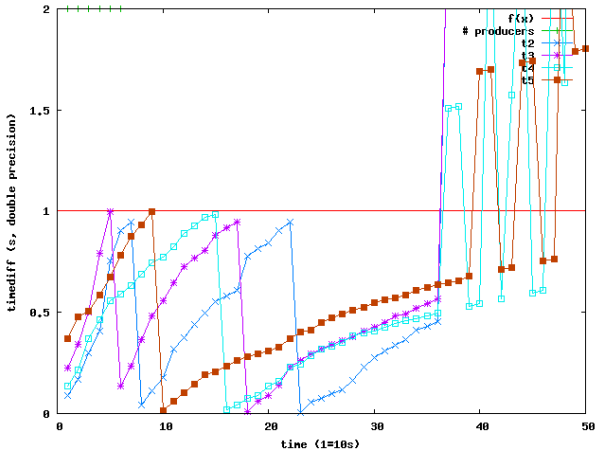


Abbildung 9.5: Zeitdifferenz ist ab 12 Vehikel für t2 und t3 nicht mehr unter 1 Sekunde haltbar, t4 und t5 schaffen es manchmal bei 14 Vehikeln.

Bibliotheken gefunden, was aber durch Korrektur einiger Header imports gelöst werden konnte. Der resultierende diff-patch befindet sich ebenfalls auf dem Datenträger, so wie auch ein startproc/rc init Skript für den safmq daemon.

Mindestens 1 Message-Queues wird benötigt:
Die *query_queue*, für den Empfang von Nachrichten. Wie in Croquets Islands haben wir nur ein Listener-Thread in der VR für die query_queue, was die Koordination deutlich vereinfacht.

Für jede Message-Queue müssen Username und Passwort vergeben werden. In Testumgebungen kann der admin user verwendet werden, der per default Zugriff auf alle Queues besitzt.

Die mitgelieferte VR Umgebung kann per Skript gestartet werden.
Für Linux x86:

```
main.sh
```

Für Windows x86:

```
main.bat
```

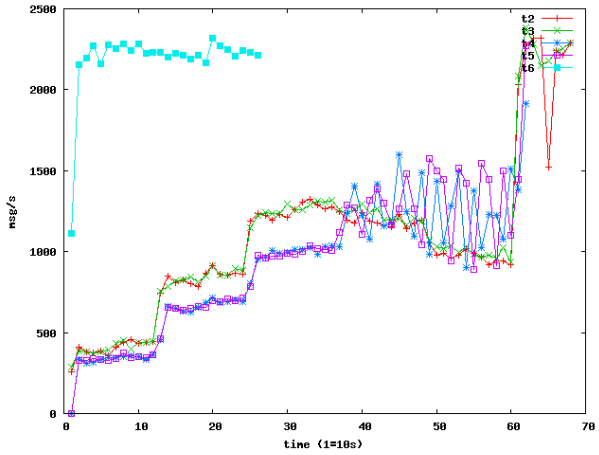


Abbildung 9.6: Nachrichtedurchsatz pro Vehikel. Der Sprung ab t=60 deutet das Ende der producer. t6 bleibt in dieser Hinsicht unerreichbar.

Mehr Informationen bietet das Skript selbst.

```
 D:\test>main.bat help

 vr.jar: a vr environment accesible by message queue.

Usage: ./main.sh [pt] [co [num]]

Options:
-help, h: show this help
-perftest,pt: automatic test
-clientsonly,co [num]:
  start num (random) clients only
  (dont start local environment)
  Can be combined withthe perftest option
  to do performance tests on remote environments.

To change configuration, create
a java properties file named `mrapi.properties`
in your current workdir.

Known properties are:
query_queue_uri:
        destination of client querys.
        vr listener awaits messages here.
```

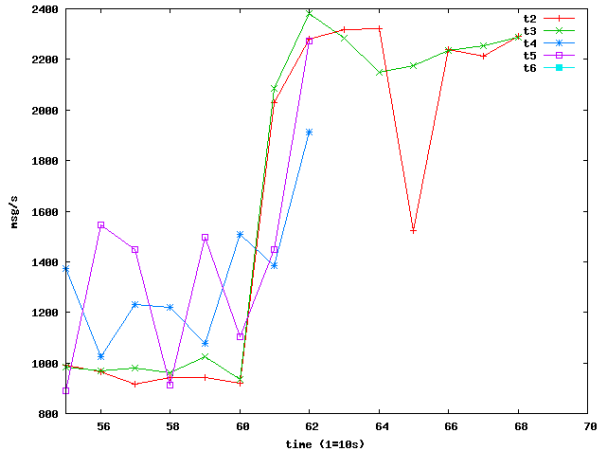


Abbildung 9.7: Zeit die die Testläufe für das Ende der Verabeitung ab t=60 benötigen. t6 wird nicht angezeigt.

```
response_queue_uri:
        destination of vr events and TTL errors.
        Clients using shared queues should take out only messages,
        that contain their clientid as recipientid.

Expected value:
  A URI with safmq protocol, userinfo, host and port.

Example:
  query_queue_uri=safmq://admin:admin@localhost:9000/query

To change map layout, create ascii file `map.buildfile` in your workdir
Fill it with `p`hysical and `s`ensor tiles.
```

Der Aufruf ohne Parameter startet eine vr.jar Instanz, die sich über einen MessageListenerThread mit der MessageQueue *query_queue_uri* verbindet. Zur Konfiguration der Anwendung kann im Arbeitsverzeichnis die Datei

```
mrapi.properties
```

angelegt werden. Darin können die URIs der beiden Message-Queue eingetragen werden. Eine URI muss aus Sicht der Applikation folgende Struktur aufweisen:

```
safmq://$user:$passwort@$host:$port/$queue
```

Erscheint ein Controlpanel (Abbildung: 9.8) und keine Fehler auf der Konsole, ist die Instanz fertig für die Verarbeitung und Darstellung der Bewegungsnachrichten an Vehikel.

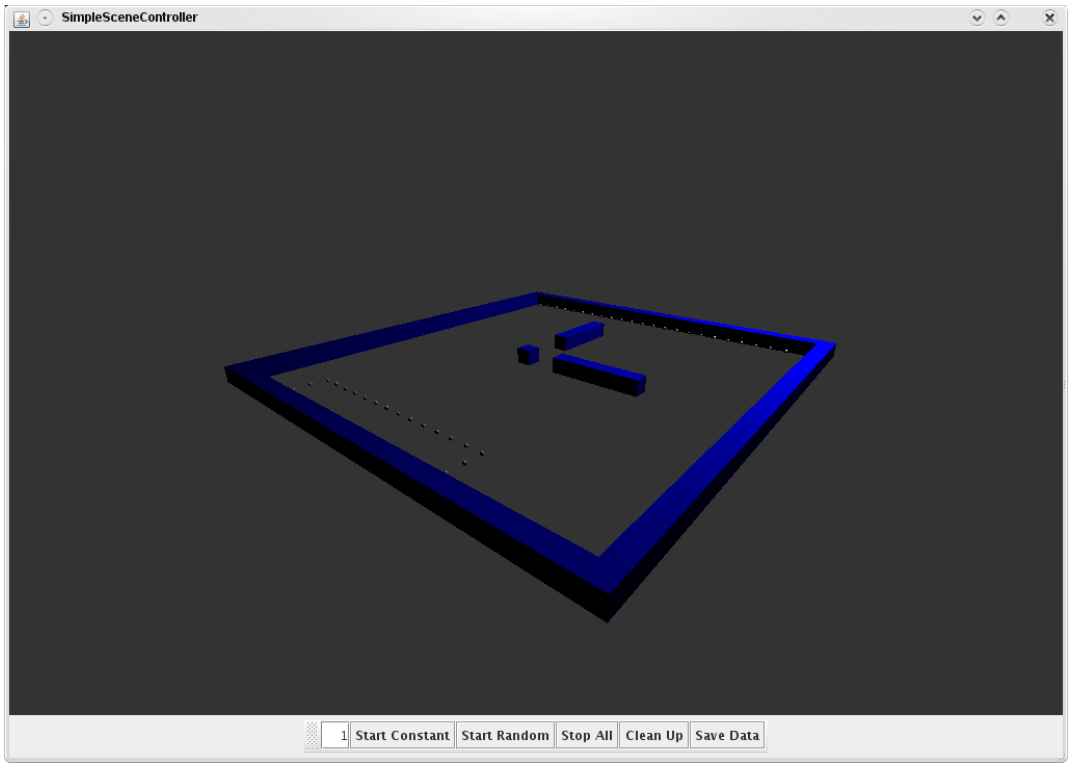


Abbildung 9.8: Test VR Umgebung

### 9.2.1 Ablauf einer Nachricht (Javaclient)

Mit Hilfe des enum LabelTokens.java werden die Bewegungsanweisungen 7.1.3 beschrieben. Es dient als Basis für den Versand und Auswertung von Nachrichten. Wir verwenden *Tokens* statt zum Beispiel einer vollständigen Grammatik, a) wegen der geringen Anzahl an möglichen Befehlen und b) weil schon beim schreiben des Codes überprüft werden kann ob das Token existiert Nachrichten von nicht Java Systeme gehen als String ein und werden zu einem EnumSet<LabelTokens> mit Hilfe der Methode LabelToken.setOF(String... strings) gepackt. Zunächst aber die kurze Fassung der Klasse:

```
public enum LabelTokens
{
    vehicle,
    forwardZ,
    backwardZ,
    anticlockwiseY,
    clockwiseY,
    setposition
}
```

Ein Java Client oder Jadexagent benutzt die LabelTokens beim Aufruf des Moveable Interface (VehicleRemoteController)

```
vehicle.getMoveable().perform( forwardZ );
```

Es können auch mehrere Bewegungen angegeben werden, in der Form

```
vehicle.getMoveable().perform( forwardZ, forwardY, clockwiseY, ... );
```

Die Fernbedienung verfügt über einen MessagingClient zum Versand der Nachricht. Die resultiernde Nachricht ist:

```
"clientidUUID|vehicle forwardZ, forwardY, clockwiseY"
```

Die ClientID wird vom MessagingClient mit dem SAFMQ Server ausgehandelt (9.2.2).

Nach Versand wir die Nachricht wird vom ListenerThread der VR aufgegriffen. Der ListenerThread liefert die Nachricht bei der VR in Form eines MessageHolder an. Daraus kann die VR die ClientID und die LabelTokens beziehen. Die VR interessiert sich zunächst nur für den *vehicle* Part der Nachricht, was sie durch die Operation

```
if(tokens.remove(vehicle))
```

ausdrückt. Ist das *vehicle* Token nicht vorhanden, verwirft die VR die Nachricht. Bei Vorhandensein des tokens wird dieser entfernt und die Bedingung erfüllt. Anhand der ClientID kann die VR nun das Vehikel in der virtuellen Welt zuordnen und diesem die restliche LabelTokens an die handle Methode übergeben.

```
//Vehicle
public void handle( EnumSet<LabelTokens> tokens )
{
        if ( getBody() != null && getBody().isSimEnabled() )
        {
                getBody().evaluate( tokens, body );

        } else if ( getMoveable() != null )
        {
                getMoveable().perform( tokens, body );

        }
}
```

```
//Moveable
public void perform( EnumSet<LabelTokens> tokens )
{
for ( LabelTokens token : tokens )
{
        if ( LabelTokens.Movements().contains( token ) )
        {
                move( token );

        } else if ( LabelTokens.Rotations().contains( token ) )
        {
                rotate( token );
        }
}

//Rotation
private void rotate( LabelTokens... direction )
{
for ( LabelTokens token : direction )
    {
        if ( token == anticlockwiseY )
        {
          Transform3D rotateAntiClockWise = new Transform3D();
          rotateAntiClockWise.rotY( ( float ) Math.PI / 4f );
          getTransform().mul( rotateAntiClockWise );
        } else if ( token == clockwiseY )
        {
          Transform3D rotateClockwise = new Transform3D();
          rotateClockwise.rotY( ( float ) ((Math.PI / 4f) * -1) );
          getTransform().mul( rotateClockwise );
        }
    }
}
```

### 9.2.2 ClientID aushandeln mit SAFMQ

Ein zuverlässiges Mechanismus zum Erhalt einer eindeutigen ID ist das Holen einer UUID direkt vom SAFMQ Server. Dazu schickt der Client eine Nachricht mit beliebigem Label (ohne Body) an den Server. Bei Annahme beinhaltet die Antwort des Servers nebem dem

SAFMQ.ErrorCode (im Normallfall 0) auch die UUID der Nachricht. Diese wird im nativen Client auf stdout gezeigt. Bei den Interfaces erhält man die UUID vom MessageObjekt mit der Methode getMessageID(). Diese UUID kann im folgenden als ClientID verwendet werden.

### 9.2.3 Ablauf einer Nachricht (C Client)

Ein System benötigt zur Teilnahme an der VR mindestens den nativen SAFMQ client. Der Sourcecode für die Kompilierung wurde dem Datenträger beigelegt, alternativ kann es online[79] bezogen werden. Der Windows Client ist im Installationpaket enthalten. Der Linux Client wird durch den Befehl

```
./configure
make all
```

mitkompiliert. Siehe auch Hinweise zur Kompilation in 9.2.

Alternativ zum nativen Client stehen das C++, Java, .NET und PHP Interface zur Verfügung. Der native Client dürfte vor allem für Roboter Interessant sein, die über keine JVM verfügen, eines der Gründe warum wir auf JMS Implementationen verzichten. Ein Roboter sollte in der Regel aber nicht Bewegungsnachrichten schicken, es sein denn ein Vehikel wurde implementiert der die Bewegung 1:1 virtuell nachmachen kann. In der Regel wird ein ART oder anderes Positionierungsystem die aktuelle Position des Roboters mitteilen.

Als Stellvertreter für ein solches System nehmen wir hier ein simples bash Skript. Der Roboter hat die Position geändert. Um diese den System mitzuteilen, wird die folgende Nachricht gesendet:

```
echo -e x=$X\n\
        y=$Y\n\
        z=$Z |\
safmqc --enqueue --label="$myid|vehicle_mockup setposition X Y Z" $VrS
```

#### 9.2.3.1 Erläuterung

```
echo -e x=$X\n\
        y=$Y\n\
        z=$Z |\
```

[79] http://safmq.sourceforge.com/

Der Inhalt der Nachricht (Body) ist die neue Position. Die VR Anwendung ist in Java, und nimmt kann neben Inpustreams praktischerweise auch gleich Properties aus dem Body herausholen. Also wird die Position als Schlüssel/Wert Tupel, getrennt von = und gefolgt von n(wewline) angegeben. Der Inhalt wird per pipe in safmqc stdin geleitet.

```
safmqc --enqueue --label="$myid|vehicle_mockup setposition X Y Z" $VrSa
```

Stellt eine Nachricht in die Queue unter der Adresse $VrSafmqURI, hier als Variable definiert. Die Adresse erfährt man vom Betreiber der VR Seite. Der Inhalt der Nachricht ist stdin.

Die Nachricht enthält den Titel (Label):

```
$myid|vehicle_mockup setposition X Y Z
```

Die Variable $myid enthält die eindeutige ID dieses Clients. Es kann eine beliebige ID genommen werden, aber das kann die Simulation durcheinander bringen, wenn ein anderer Client die selbe ID wählt. Wie eine eindeutige ClientID bezogen wird steht in 9.2.2.

*vehicle_mockup* ist das vereinbarte Token für Vehikel Attrappen 7.1.4 in der Simulation. *setpostion X Y Z* gibt der VR den Befehl die Position der Attrappe anhand der Koordinaten zu setzen. Die angegebenen Koordinaten verraten der VR nach welchen Schlüsseln im Body gesucht werden soll (Deswegen auch das Java Properties Format).

#### 9.2.3.2 Ablauf eines Ereignisses

Der Client gibt eine Antwortadresse in einer seiner Nachricht an. Das System ordnet der ClientID die Anwortadresse zu. Tritt ein Ereignis auf, z.B. über ein CollisionListener, bekommt das System die beteiligten ClientIDs und Daten der Ereignisauswertung vom Simulationsbackend. Das System reiht eine Nachricht mit den Daten in die Message-Queue unter der zuletzt bekannten Antwortadresse des Clients ein.

Der Client sollte einen nebenläufigen MessageListener gestartet haben. Wird die Queue mit anderen Clients geteilt, sollte der MessageListener nur Nachrichten raus holen deren RecipientID die eigene ClientID ist. Der Client kann aber die Adresse einer eigenen Message-Queue geben. Es wird empfohlen, den Client auf der selben Maschine zu betreiben von der er Nachrichten abholt.

## 9.3 Systemdokumentation

Der Sourcecode der Projekte wird auf dem Datenträger mitgeliefert. Die in Java implementiertes Teile des Systems wurden in folgende Teilprojekte unterteilt:

1. Commons Bindeglied zwischen den Projekten. Enthält das Anwendungsmodel und Kommunikationsmodul.
2. JadexAgent Enthält ein einfaches ADF samt Plan zur Ausführung zufälliger Bewegungen durch ein Vehikel.
3. Vehicles Transportmittel für Agenten
4. VirtualReality Visualisierung und Simulation

### 9.3.1 Commons

Enthält die Pakete:

enums: Bekannte, sich in message labels oft wiederholende *tokens* (befehlsschnippsel) und andere feste Eigenschaften werden in *Enum Types* (Esser, 2005) gepflegt.

model: Hierin sind die Interfaces (Schnittstellen) zwischen VR Umgebung und Vehikel modelliert.

messaging Alles was wird bei nachrichtenorientierte Kommunikation über SAFMQ verwenden ist hierin enthalten.

#### 9.3.1.1 Klassenbaum

```
Commons/src/
`-- mrapi
    |-- Config.java
    |-- enums
    |   |-- BasicShapes.java
    |   |-- GraphicsEngines.java
    |   |-- LabelTokens.java
    |   `-- Tiles.java
    |-- messaging
    |   |-- MessageHolder.java
    |   |-- MessagingClient.java
    |   |-- MessagingException.java
```

```
|   |-- MessagingListener.java
|   |-- MessagingTopicPublisher.java
|   `-- runnables
|       |-- RunnableListener.java
|       `-- TopicPublisher.java
`-- model
    |-- Driver.java
    |-- Graphics.java
    |-- ListeningMessageHandler.java
    |-- Publisher.java
    |-- Simulation.java
    |-- Vehicle.java
    `-- deeper
        |-- AbstractVehicle.java
        |-- Angle.java
        |-- Body.java
        |-- Distance.java
        |-- Event.java
        |-- Moveable.java
        |-- Shape.java
        |-- ValueHolder.java
        |-- ValueHolderAbstract.java
        `-- Vector3N.java
```

### 9.3.2 JadexAgent

Beinhaltet die Implementierung eines JadexAgenten, der ein Vehikel in der VR Umgebung per Zufall steuert.

#### 9.3.2.1 Klassenbaum

```
./JadexAgent/src
|-- jadex
|   `-- examples
|       `-- dmr
|          |-- DumbVehicleMover.agent.xml
|          |-- MultipleStarter.agent.xml
|          |-- SimpleVehiclePatrolPlan.java
|          `-- StartAgentsPlan.java
`-- test
```

```
    `-- Main.java
```

### 9.3.3 Vehicles:

Eine neue Vehikelklasse wird durch Implementierung des Vehicle Interfaces erstellt. Enthaltene Pakete:

impl Beinhaltet Vehikelimplementierungen. Die Implementation der einzelnen Bestandteile eines Vehikels sind sehr stark an die Visualsierungs- und Simulationsimplemeniterung gekoppelt, weshalb das Vehicle Interface (in Commons 9.3.1) Generische Datentypen (Esser, 2005) verwendet.

test Klassen mit statischen Testmethoden.

#### 9.3.3.1 Klassenbaum

```
./Vehicles/src
`-- mrapi
    |-- impl
    |   |-- drivers
    |   |   `-- DefaultDriver.java
    |   `-- vehicles
    |       |-- AgentVehicleStub.java
    |       |-- DefaultVehicle.java
    |       `-- VehicleRemoteController.java
    `-- test
        `-- TestVehicle.java
```

### 9.3.4 VirtualReality

Beinhaltet Implementierung einer virtuellen Umgebung und Nachrichtenverarbeitung. Die Visualisierung ist in Xith3D implementiert. Die Implementierung der Simulationskomponente war bis zu Fertigstellung der schriftlichen Fassung aus zeitlichen Gründen nicht möglich. Die Startskripte rufen die main Methode der hier der enthalten Main Klasse [80] auf.

#### 9.3.4.1 Klassenbaum

```
./VirtualReality/src
|-- map.buildfile
```

[80] Ein guter Ort für den ersten Breakpoint

```
|-- mrapi
|   |-- impl
|   |   |-- drivers
|   |   |   `-- RandomDriver.java
|   |   `-- xith3d
|   |       |-- Xith3dGraphics.java
|   |       |-- Xith3dMover.java
|   |       |-- Xith3dScene.java
|   |       `-- Xith3dShape.java
|   `-- test
|       |-- Main.java
|       |-- MapNotFound.java
|       |-- SimpleSceneController.java
|       |-- TerrainMap.java
|       |-- TerrainMapReader.java
|       |-- TestEnvironmentPerformance.java
|       `-- TestPerformance.java
`-- mrapi.properties
```

# Teil III

# Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde die Machbarkeit eines Systems untersucht, das neben der Darstellung realer und virtueller Roboter in einer virtuellen Umgebung, das Zusammenspiel der Akteure ermöglicht. Zunächst wurden die Konzepte der drei dimensionalen Darstellung auf einem Bildschirm und die Erweiterung einer virtuellen Umgebung um simulierte Körperdynamik untersucht.

Das Konzept eines Vehikels wurde aufgegriffen, um es inbesondere Agenten zu erleichtern, sich virtuell auf die Steuerung eines Roboters in der realen Umgebung vorzubereiten. Dabei wurde der fließende Übergang zwischen den Realitäten berücksichtigt: Agenten können Roboter in der Realität steuern, dabei aber der Lösung von Aufgaben in der virtuellen Umgebung nachgehen. Ein weiterer Aspekt bei der Implementierung eines Vehikels war die Beschreibung und Implementierung einer einfachen Kontrollstruktur. Diese erlaubt es dem Entwickler von Agentenplänen, Bewegungsabläufe des Vehikels im 3-D Raum zu bestimmen, ohne die dafür notwendigen Transformationen im Detail zu kennen. Dieses Konzept soll bei der Erweiterung der Fähigkeiten eines Vehikels beibehalten werden. Ein Ziel ist es Wege zu finden, die Kontrollstruktur so zu gestalten, dass Agenten damit selbständig heraus finden können, wozu ein Vehikel in der Lage ist.

Dem Kommunikationsprotokoll wurde besondere Aufmerksamkeit geschenkt. Die anfängliche Defnition einer Schnittstelle zwischen einer generischen, virtuellen Umgebung und einem Roboter gestaltete sich schwierig. Daher wurden lose gekoppelte, nachrichtenorientierte Kommunikationsmodelle untersucht. Dies geschah unter den Gesichtspunkten flexibler Nachrichteninhalte, Sequentialisierung der Nachrichten und garantierter Auslieferung simulationrelevanter Daten. Durch den Einsatz von Message-Queues konnten diese Bereiche zufriedenstellend abgedeckt werden. Das System ermöglicht dadurch die Durchführung zweier Arten von Simulation:

Passiv: Die zeitlich entkoppelte Übermittlung von Positions- und Bewegungsdaten zur späteren Auswertung und Visualisierung.

Aktiv: Die Anlieferung und Auswertung in Echtzeit.

Anhand von Messungen konnte gezeigt werden, dass das System bei bis zu 10 Teilnehmern weiche Echtzeitanforderungen bis 1 Sekunde erfüllen kann. Der für die Machbarkeitstudie und Messungen implementierte Prototyp soll auch als Basis für eine Weiterentwicklung des Systems dienen.

Die Skalierbarkeit der zunächst zentralen VR ist ein wunder Punkt, der sich durch Verteilung lösen läßt. Ein Verteilungskonzept ähnlich den Regions von OpenSimulator oder Croquet Islands wäre erstrebenswert. Des weiteren soll die Fähigkeiten der Vehikel deutlich erweitert werden. Durch den Entwurf und Einbezug von austauschbaren Sensoren und Aktoren sollen dem Agenten durch das Vehikel Möglichkeiten geboten werden die Umgebung zu manipulieren oder aktiv zu gestalten.

# Literaturverzeichnis


[Benford u. a. 1998] BENFORD, Steve ; GREENHALGH, Chris ; REYNARD, Gail ; BROWN, Chris ; KOLEVA, Boriana: Understanding and constructing shared spaces with mixed-reality boundaries. In: *ACM Trans. Comput.-Hum. Interact.* 5 (1998), Nr. 3, S. 185–223. – ISSN 1073-0516

[Burka 2007] BURKA, Florian: *Kameragestutzte Objektverfolgung in Echtzeit im Kontext mobiler Roboter*. Online. März 2007. – URL `http://users.informatik.haw-hamburg.de/~kvl/burka/bachelor.pdf`

[Buss 2003] BUSS, Samuel R.: *3D Computer Graphics: a mathematical introduction with OpenGL*. Cambridge University Press, 2003. – ISBN 0521821037 hardback

[Böhm 2002] BÖHM, Oliver: *Java Software Engineering unter Linux*. SuSE-PRESS, 2002. – ISBN 3-935922-53-1

[Davison 2005] DAVISON, Andrew: *Killer Game Programming in Java*. O'Reilly Media, Inc., May 2005. – ISBN 0596007302

[Esser 2005] ESSER, Friedrich: *Java 5 im Einsatz*. Galileo Press GmbH, 2005. – ISBN 3898424596

[Grand 2002] GRAND, Mark: *Java Enterprise Design Patterns*. Bd. Patterns in Java Volume 3. Wiley Computer Publishing, 2002

[Hampshire u. a. 2006] HAMPSHIRE, Alastair ; SEICHTER, Hartmut ; GRASSET, Raphaël ; BILLINGHURST, Mark: Augmented Reality Authoring: Generic Context from Programmer to Designer. In: *OZCHI '06: Proceedings of the 20th conference of the computer-human interaction special interest group (CHISIG) of Australia on Computer-human interaction: design: activities, artefacts and environments*. New York, NY, USA : ACM, 2006, S. 409–412. – ISBN 1-59593-545-2

[Helmchen 1995] HELMCHEN, Dipl.-Päd. Dipl.-Ing. G.: Technologiegestützte Ausbildungssysteme an der Mensch/Maschine-Schnittstelle. In: *Simulation und Simulatoren - Mobilität virtuell gestalten* Ottobrunn (Veranst.), VDI-Verlag, 1995

[Kahlbrandt 2005] KAHLBRANDT, Bernd: *Software-Engineering*. 2005

[Kirchknopf und Witta 2003] KIRCHKNOPF, Dr.-Ing. P. ; WITTA, Dr.-Ing. L.: Der neue BMW - Schwingungs- und Akustiksimulator, Aufbau und Anwendungen. In: *Simulation und Simulatoren - Mobilität virtuell gestalten* BMW Group, München (Veranst.), VDI-Verlag, 2003

[Koester 2008] KOESTER, Frank: *Datenmanagement als Rueckgrat einer menschzentrierten Entwicklung von Fahrerassistenzsystemen*. March 2008. – Cebit 2008 „Forum for Telematics & Navigation, Automotive Solutions, Transport & Logistics“

[Lindinger u. a. 2006] LINDINGER, Christopher ; HARING, Roland ; HÖRTNER, Horst ; KUKA, Daniela ; KATO, Hirokazu: Multi-user mixed reality system; Gullivers World: a case study on collaborative edutainment at the intersection of material and virtual worlds. In: *Virtual Real.* 10 (2006), Nr. 2, S. 109–118. – ISSN 1359-4338

[Manojlovich u. a. 2003] MANOJLOVICH, Joseph ; PRASITHSANGAREE, Phongsak ; HUGHES, Stephen ; CHEN, Jinlin ; LEWIS, Michael ; AVE, N. B.: Utsaf: A multi-agent-based framework for supporting military-based distributed interactive simulations in 3d virtual environments. In: *in 3D Virtual Environments, Proceedings of the Winter Simulation Conference*, Press, 2003, S. 960–968

[Milgram und Kishino 1994] MILGRAM, P. ; KISHINO, F.: A taxonomy of mixed reality visual displays. IEICE Trans. Inf. Syst. E77-D (Veranst.), Dec 1994

[Millington 2007] MILLINGTON, Ian: *Game Physics Engine Development*. Morgan Kaufmann Publishers, Elsevier Inc., 2007. – ISBN 012369471X

[Pareigis u. a. 2007] PAREIGIS, Stephan ; SCHWARZ, Bernd ; KORF, Franz: *FAUST: Entwicklung von Fahrerassistenz- und autonomen Systemen*. Kap. 4, S. 69–78, Springer, Informatik Aktuell, November 2007

[Preissler 1995] PREISSLER, Dipl.-Phys. H.: Achtung Stromausfall! - Von simulierten Wirklichkeiten und realen Illusionen. In: *Simulation und Simulatoren - Mobilität virtuell gestalten* Berlin (Veranst.), VDI-Verlag, 1995

[Raasch 1991] RAASCH, Jörg: *Systementwicklung mit Strukturierten Methoden: ein Letfaden für Praxis und Studium*. München; Wien : Hanser, 1991. – 333–388 S. – ISBN 3-446-16147-3

[Richter 2008] RICHTER, Prof. Dr.-Ing. H.: *Car Ring II - Echtzeitrechnernetz für die Intra-Automobil-Kommunikation*. 2008. – URL `http://www.in.tu-clausthal.de/abteilungen/technische-informatik-und-rechnersysteme/forschung/abteilung-rechnernetze/projekt-carring-ii/`

[Russel und Norvig 2003] RUSSEL, S. ; NORVIG, P.: *Artificial Intelligence: A Modern Approach*. Second Edition, Prentice Hall. 2003

[Sabeur 2003] SABEUR, Dr.-Ing. M.: 3D basierendes "Virtual Prototyping"von Mensch-Maschine Schnittstellen, ein neuer Ansatz zur Optimierung des Entwicklungsprozesses von Cockpitsystemen. In: *Simulation und Simulatoren - Mobilität virtuell gestalten* Princess Interactive Software GmbH, Magdeburg (Veranst.), VDI-Verlag, 2003

[Sloan 2005] SLOAN, Joseph D.: *High Performance Linux Clusters with OSCAR, Rocks, openMosix, and MPI*. O'Reilly, 2005. – ISBN 0-596-00570-9

[Tanenbaum und van Steen 2008] TANENBAUM, Andrew ; STEEN, Maarten van: *Verteilte Systeme. Grundlagen und Paradigmen*. 2., aktualisierte Aufl. Pearson Studium, 2008. – ISBN 3-8273-7293-3, 978-3-8273-7293-2

[Tso u. a. 1999] TSO, K.S. ; THARP, G.K. ; ZHANG, W. ; TAI, A.T.: A multi-agent operator interface for unmanned aerial vehicles. In: *Digital Avionics Systems Conference, 1999. Proceedings. 18th* Bd. 2. St Louis, MO, USA, 10 1999, S. 6.A.4–1–6.A.4–8 vol.2. – ISBN: 0-7803-5749-3

[Wooldridge 2002] WOOLDRIDGE, Michael: *An introduction to multiagent systems*. John Wiley & Sons, June 2002. – ISBN 047149691X

*Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.*

Hamburg, 10. November 2008 Christoph Hentschel