Semantische Modellierung virtueller Umgebungen auf Basis einer modularen Simulationsarchitektur

Dissertation
eingereicht an der Technischen Fakultät der Universität Bielefeld
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
(Dr.-Ing.)

von
Christian Fröhlich

Universität Bielefeld, Technische Fakultät
Universitätsstr. 25
33615 Bielefeld

Semantische Modellierung virtueller Umgebungen auf Basis einer modularen Simulationsarchitektur

Christian Fröhlich
AG Wissensbasierte Systeme (Künstliche Intelligenz)
Technische Fakultät
Universität Bielefeld
Postfach 10 01 31
D-33501 Bielefeld
Germany
E-Mail: cfroehli@techfak.uni-bielefeld.de

Von der Technischen Fakultät der Universität Bielefeld zur Erlangung des Grades eines Doktors der Ingenieurwissenschaften (Dr.-Ing.) genehmigte Dissertation.

Dekan der Fakultät:
Prof. Dr. Mario Botsch

Gutachter:
Prof. Dr. Ipke Wachsmuth
Prof. Dr.-Ing. Ulrich Rückert

Einreichung der Arbeit: 29. April 2014
Tag der Verteidigung: 17. Juli 2014

Die offizielle Druckversion wurde auf alterungsbeständigem Papier gemäß DIN-ISO 9706 gedruckt.

Danksagungen

Als erstes möchte ich meinem Doktorvater Herrn Prof. Dr. Ipke Wachsmuth danken: Lieber Ipke, vielen Dank für das entgegengebrachte Vertrauen und vor allem dafür, dass Du immer an mich geglaubt hast, insbesondere in schwierigen Zeiten! Du hast mir diese Chance eröffnet und diese Promotion erst möglich gemacht. Es war mir wahrlich eine Freude mich in diesem spannenden Umfeld bewegen zu dürfen und Teil einer wundervollen Arbeitsgruppe unter Deiner Leitung gewesen zu sein. Ein sehr herzlicher Dank geht an Herrn Prof. Dr.-Ing. Ulrich Rückert, für die sehr kurzfristige Bereitschaft ein Zweitgutachten anzufertigen. Ich weiß diesen Einsatz sehr zu schätzen! Ich danke außerdem Herrn Prof. Dr. Jens Stoye und Herrn Dr. Carsten Gnörlich für das Mitwirken in der Prüfungskommission.

Bei meinen vielen Kollegen aus der AG WBS bedanke ich mich für eine wundervolle Zeit. Die konstruktive Zusammenarbeit in einer freundschaftlichen Arbeitsatmosphäre hat nicht nur zum Gelingen dieser Arbeit beigetragen; sie war mir auch stets eine Freude. Gesonderter Dank geht vor allem an Peter Biermann. Auch meinem Arbeitgeber und meinen Kollegen bei der Schüco Service GmbH danke ich für die Unterstützung und das Verständnis.

Bei meinen Freunden möchte ich mich für die moralische Unterstützung bedanken. Dass ich euch alle nach so vielen Jahren noch immer meine Freunde nennen darf, macht mich stolz. Danke an meine Eltern und meinen Bruder für die Unterstützung in jeglicher Hinsicht. Ihr habt mir vieles im Leben ermöglicht und ohne euch wäre diese Arbeit nicht zu Stande gekommen. Danke auch an meine Großmutter, die sicher stolz gewesen wäre.

Zu guter Letzt geht mein Dank an Dich Julia: Du bist ohne jeden Zweifel die beste Ehefrau, Lebenspartnerin und Freundin, die ich mir vorstellen kann. Ich schätze mich glücklich Dich in meinem Leben zu haben, da Du es jeden Tag erneut bereicherst. Du hast mich immer wieder auf ein Neues motiviert wenn ich aufgeben wollte, mich bedingungslos in Allem unterstützt was ich getan habe und mir in den schwersten Momenten wieder auf die Beine geholfen. Meinen unendlichen Dank dafür kann ich nicht weiter in Worte fassen und sage einfach: Danke, ich liebe Dich!

INHALTSVERZEICHNIS

Abbildungsverzeichnis **v**

Abbildungsverzeichnis **v**

Listingverzeichnis **vii**

1 Einleitung **1**

1.1 Motivation 3
1.2 Zielsetzung und Fokus der Arbeit 6
1.3 Forschungskontext 8
1.4 Aufbau der vorliegenden Arbeit 10
1.5 Typographische Konventionen 12

2 Simulationssysteme für virtuelle Umgebungen **13**

2.1 Monolithische Ansätze 19
2.1.1 Avango 19
2.1.2 OpenSG 21
2.1.3 OpenSceneGraph 24
2.1.4 InstantReality 27
2.2 Modulare Ansätze 29
2.2.1 FlowVR 29
2.2.2 Lightning 33
2.3 Actor Model 35
2.3.1 Direkte Kommunikation und Asynchronität 37

2.3.2 Variable Topologie 38
2.3.3 Reihenfolge der Nachrichtenverarbeitung 38
2.3.4 Heutige Relevanz 39
2.4 Zusammenfassung 41

3 Grundlagen semantischer Modellierung 43
3.1 Semantische Anreicherung 45
3.1.1 Virtuelle Konstruktion 47
3.1.2 Anwendung im multimodalen Bereich 49
3.2 Semantische Reflexion 51
3.3 Semantische Netze als Wissensrepräsentation 53
3.4 Semantische Traverser 56
3.5 Allens Intervallalgebra 58
3.6 Zusammenfassung 60

4 Konzept und Integration 63
4.1 Technische Ausgangsbasis 64
4.1.1 Kontrollfluss 69
4.2 Modulare Simulationsarchitektur 72
4.2.1 Anforderungen 72
4.2.2 Konzept 73
4.2.3 Exemplarische Integration 75
4.2.3.1 Integration des Actor Models in SCIVE . 75
4.2.3.2 Semantische Reflexion in SCIVE 81
4.3 Semantische Modellierung 86
4.3.1 Anforderungen 87
4.3.2 Konzept 88
4.3.3 Exemplarische Integration 90
4.3.3.1 Virtuelle Inhalte 91
4.3.3.2 Semantische Traverser 93
4.3.3.3 Interaktiv generierte Objekte 95
4.3.3.4 Funktionen 97
4.3.3.5 Zeitliche Abfolgen 99

4.4 Zusammenfassung 101

5 Implementierung **103**
5.1 Erweiterung der Architektur 104
5.1.1 Modularisierung der Skriptschnittstelle 104
5.1.2 Erweiterung um eine Skript-Komponente 105
5.2 Semantische Modellierung 109
5.2.1 Virtuelle Inhalte 110
5.2.2 Systemkonfigurationen 115
5.2.3 Ikonische Formen und Gesten 117
5.2.4 Funktionen 120
5.2.5 Temporale Sequenzen 124
5.3 Zusammenfassung 128

6 Anwendungsbeispiel **129**
6.1 Technische Umgebung 130
6.2 Virtuelle Konstruktion 132
6.2.1 Wissensbasierte Computergrafik 133
6.2.2 Modularisierung und semantische Modellierung 136
6.3 Zusammenfassung 139

7 Resümee **141**
7.1 Eine modulare Simulationsarchitektur 142
7.1.1 Architektur 142
7.1.1.1 Bewertung 143
7.1.2 Integration von VR-Methoden 144
7.1.2.1 Bewertung 145
7.2 Abstraktion der Simulationsinhalte und Anwendungslogik . . . 146
7.2.1 Bewertung 147
7.3 Eine abschließende Betrachtung 148

Literaturverzeichnis **151**

ABBILDUNGSVERZEICHNIS

1.1 Eine längere Persistenz durch Abstraktion und Modularität . . 6

2.1 Sensorama und eine Abbildung aus dem zugehörigen Patent 15
2.2 Schema einer CAVE und exemplarische Anwendung 16
2.3 Illustration einer monolithischen Simulationsarchitektur 17
2.4 Illustration einer modularen Simulationsarchitektur 18
2.5 Schichten der Avango Architektur 19
2.6 Cluster-Setup mit zugehöriger Ausgabe in OpenSG 23
2.7 Schichten der OpenSceneGraph Architektur 25
2.8 Schichten der InstantReality Architektur 27
2.9 Beispielhafte Illustration eines FlowVR Netzwerkes 32
2.10 Schichten der Lightning Architektur 34
2.11 Illustration der Funktionsweise des Actor Models 37

3.1 Partiell semantisch angereicherter Szenengraph 46
3.2 Ein semantisches Netz 54
3.3 Ein funktional erweiterbares semantisches Netz 55
3.4 Beispielhafte Traversierung eines Szenengraphen 57
3.5 Allens 13 zeitliche Relationen im Überblick 58
3.6 Beispielhafte Anwendung des Allen-Kalküls 60

4.1 Simulationsarchitektur mit fünf Modulen 64
4.2 Simulationsarchitektur mit zentraler Organisation 67
4.3 Simulationsarchitektur mit dazugehörigem Datenaustausch . . 68

4.4 Beispiel des Kontrollflusses in SCIVE 71
4.5 Konzept der modularen Simulationsarchitektur 74
4.6 Nachrichtenaustausch in SCIVE 77
4.7 Kapselung eines Moduls in eine Aktor-Schicht 78
4.8 Anwendung der semantischen Reflexion 81
4.9 Integration verschiedener Funktion in die Grafikkomponente . 83
4.10 Reflexion von Funktionalitäten in der Wissensrepräsentation . 84
4.11 Konzept der einheitlichen Wissensrepräsentation 88
4.12 Domänen im semantischen Netz 91
4.13 Modellierung virtueller Objekte 92
4.14 Traverser-Manager 93
4.15 Modellierung eines semantischen Traversers 94
4.16 Ikonische Geste mit begleitender verbaler Instruktion 96
4.17 Interaktive Generierung von Bauteilen 97
4.18 Semantische Reflexion einer C++ Funktion 98
4.19 Semantische Reflexion einer temporalen Sequenz 100

5.1 Übertragung der Form eines Rohres auf das semantische Netz 118
5.2 Beispielhafte semantische Definition einer Funktion 120
5.3 Beispielhafte semantische Definition einer temporalen Sequenz 125

6.1 Technische Umgebung der AG WBS im Überblick 131
6.2 Beispielhafte Szene aus der CAVE der AG WBS 132
6.3 Zweihändige Skalierung in der Virtuellen Werkstatt 133
6.4 Unterschiedliche gekrümmte Formen und Krümmungswinkel . 134
6.5 Formbeschreibung auf semantischer Ebene 135
6.6 Datenhaltung im Überblick 138

LISTINGVERZEICHNIS

2.1 Definition einer dreidimensionalen Box in X3D 28
5.1 Zugriff auf verschiedene Module mittels Python 107
5.2 Zugriff auf die zentrale Datenbasis in Python 109
5.3 Szenendefinition auf Basis des semantischen Netzes 110
5.4 In Domänen aufgeteiltes semantisches Netz 111
5.5 Semantisches Netz mit Relationen 111
5.6 Semantisches Netz mit spezialisierten Knoten 112
5.7 Semantisches Netz mit spezialisierten Knoten und Relationen . 114
5.8 Domäne und Relationen einer Systemkonfiguration 115
5.9 Erweiterung der Systemkonfiguration um Knoten 116
5.10 Semantisches Netz zur Definition einer Systemkonfiguration . 116
5.11 Semantisches Netz zur Definition einer ikonischen Geste 118
5.12 Semantische Definition allgemeiner Funktions-Knoten 121
5.13 Semantische Definition einer konkreten C++ Funktion 121
5.14 Semantische Definition von Parametern einer C++ Funktion . 122
5.15 Semantisches Netz zur Definition eines Funktionsaufrufes . . . 123
5.16 Semantische Definition von Knoten einer temporalen Sequenz 126
5.17 Semantische Definition von temporalen Relationen nach Allen 127
5.18 Semantische Verbindung von Knoten in einer Sequenz 127

KAPITEL

1

EINLEITUNG

"A virtual world is the content of a given medium. It may exist solely in the mind of its originator or be broadcast in such a way that it can be shared with others. A virtual world can exist without being displayed in a virtual reality system (i.e., an integrated collection of hardware, software, and content assembled for producing virtual reality experiences) – much like play or film scripts exist independently of specific instances of their performance. Such scripts do in fact describe virtual worlds. Let's carry the analogy further. We can refer to the script of a play as merely the description of a play. When that description is brought to life via actors, stage sets, and music, we are experiencing the play's virtual world. Similarly, a computer-based virtual world is the description of objects within a simulation. When we view that world via a system that brings those objects and interactions to us in a physically immersive, interactive presentation, we are experiencing it via virtual reality."

William R. Sherman & Alan B. Craig [Sherman u. Craig 2002]

Betrachtet man die oben stehende Analogie zu einem Theaterstück und überträgt diese in die Welt der computergenerierten Realität, wird das Drehbuch zur Wissensbasis oder auch inhaltlichen Beschreibung der virtuellen Umgebung. Die Schauspieler werden zu virtuellen Charakteren, das Bühnenbild zur virtuellen Umgebung, die Requisiten zu virtuellen Objekten, mit denen ein Anwender interagieren kann, und zu guter Letzt wird der Regisseur zum Entwickler der virtuellen Welt. Ein reales Theaterstück benötigt unter anderem eine Lokalität für seine Aufführung, im Normalfall eine Bühne. Im übertragenen Sinne dienen ein einfacher Computermonitor oder eine fortgeschrittene dreidimensionale Projektion als Ausgabemedium für die grafische virtuelle Umgebung. Um ein Theaterstück allerdings in seinem vollen Umfang betrachten zu können, müssen wir zunächst in der Lage sein, alle seine verwendeten **Modalitäten**[1] wahrnehmen zu können. Ebenso ist es auf der anderen Seite wichtig, dass diese Modalitäten generiert werden können. Ein Theaterstück braucht gegebenenfalls neben seiner Bühne und dem damit verbundenen Bühnenbild weitere Medien, wie zum Beispiel ein Orchester, um das dargebotene Stück mit Musik untermalen zu können, oder spezielle Licht-Effekte, um einen höheren Grad an Realismus zu erwirken. Die hierfür benötigten physikalischen Medien lassen sich wiederum gut auf eine virtuelle Realität, beispielsweise in Form von Lautsprechern, übertragen.

Soweit lässt sich diese Analogie sehr gut übertragen. Weniger greifbar wird der Vergleich allerdings bei der Generierung, des von dem Zuschauer als selbstverständlich hingenommenen realen Bühnenbildes oder der perfekten musikalischen Untermalung. Während diese Dinge im Realen physikalisch aufgebaut oder von einem Dirigenten abgestimmt werden können, müssen sie im Virtuellen simuliert werden. Hierfür werden verschiedene spezialisierte Medien benötigt. Um ein virtuelles Bühnenbild auf einem physikalischen Ausgabemedium (**Hardware**) darstellen zu können, benötigt es ein Simulationssystem (**Software**). Neben der grafischen Ausgabe wird auch für die anderen Modalitäten, beispielsweise der auditiven Ausgabe, ein solches Sys-

[1] **Modalität:** von lat. *modalitas* „Art und Weise, Möglichkeit, Bedingung, Ausführungsart“.

tem benötigt. In virtuellen Umgebungen müssen darüber hinaus Medien simuliert werden, welche im Realen absolut selbstverständlich sind. So bedarf es für einen maximalen Grad an Realismus noch einer Approximation der um uns herum geltenden physikalischen Gesetze. Werden diese Modalitäten vernachlässigt verliert die Simulation an Glaubwürdigkeit. Vergleichend lässt sich sagen, dass ein moderner Kinofilm viel realistischer wirkt, als ein Stummfilm aus den Anfängen der Kinozeit.

Der Wunsch virtuelle Welten umfassend simulieren zu können, leitet in den folgenden Abschnitt über, in dem näher auf diese Thematik eingegangen wird.

1.1 Motivation

Die Motivation der vorliegenden Arbeit begründet sich auf die heutzutage bestehende Notwendigkeit, in Simulationen der Virtuellen Realität einen möglichst hohen Grad an Realismus und **Präsenz**[2] zu erzeugen. Während noch vor einigen Jahren auf Grund fehlender Rechenleistung nicht solch hohe Ansprüche gestellt wurden, ist die Computertechnik heute weit genug fortgeschritten. Außerdem sind Anwender im heutigen Zeitalter an imposante computergenerierte Bilder, zum Beispiel durch Spezialeffekte in Filmen, gewöhnt und stellen ähnliche Ansprüche an virtuelle Umgebungen.

Ein möglichst hoher Grad an Realismus lässt sich nicht durch eine akkurate Simulation grafischer Effekte allein erreichen. Während in den vergangenen Jahren der Fokus auf eben diesen lag, ist heute die Integration einer breiten Palette anderer Modalitäten von Nöten. Hieraus hervorgehend besteht die Motivation ein System zu entwickeln, welches die Simulation verschiedener Modalitäten auf intelligente Art und Weise miteinander vereint. Die einzelnen Module sollen dabei möglichst unabhängig voneinander sein, damit sie je nach Bedarf durch andere ausgetauscht werden können.

[2] **Präsenz:** Bezeichnet das Eintauchen in eine virtuelle Welt mit gleichzeitigem Ausblenden der realen Umgebung.

Eine weitere Motivation dieser Arbeit resultiert daraus, dass viele Anwendungen der Virtuellen Realität unbrauchbar werden, sobald das darunter liegende Simulationssystem gewechselt wird. Die meisten Systeme stellen zu viele Abhängigkeiten zu der konkreten Applikation her, als dass diese auf ein anderes Simulationssystem übertragen werden könnten. Wird die Weiterentwicklung eines Systems eingestellt, wie es in vergangenen Jahren aus den unterschiedlichsten Gründen immer wieder passiert ist, lassen sich in der Regel auch die auf diesem System basierenden Anwendungen nicht weiterentwickeln. Es ist durchaus möglich Anwendungen noch eine gewisse Zeit über das Bestehen des Simulationssystems (oft auch als **Framework**[3] bezeichnet) hinaus weiter betreiben zu können. Jedoch schreitet die Softwareentwicklung immer noch in einem so schnellen Maße voran, dass in der Regel nicht mehr gewartete Software innerhalb einiger Monate unbrauchbar wird.

Der Wunsch nach einem Wechsel des eingesetzten Frameworks muss allerdings nicht zwangsläufig durch eine eingestellte Weiterentwicklung des Frameworks entstehen. In einigen Fällen kommt eine solche Entscheidung auch durch die Verfügbarkeit neuerer, weiter entwickelter oder einfach komfortablerer Systeme, welche eine Steigerung der Entwicklungseffizienz bedeuten, zu Stande. Das hat meistens zur Folge, dass die bis dahin bestehenden Anwendungen nicht weiter entwickelt oder gewartet werden, sei dies aus Gründen der Kosten oder der dafür benötigten Zeit. Anwendungen auf Basis verschiedener Systeme zu entwickeln, übersteigt im Regelfall die vorhandenen Kapazitäten. Wünschenswert ist an dieser Stelle also eine Möglichkeit, einzelne Komponenten leicht gegen fortschrittlichere austauschen zu können, um mit bestehenden Anwendungen auf dem neuesten Stand der Technik zu bleiben.

Die Zusammenhänge nach der eine Anwendung funktioniert, beziehungsweise abläuft, werden unter dem Begriff der **Anwendungslogik** zusammengefasst. Sie umfasst die herrschenden Prinzipien des Datenflusses innerhalb der Anwendung, temporale Zusammenhänge sowie die Definition von be-

[3] **Framework:** Ein Gerüst, das eine Anwendungsarchiktektur vorgibt.

stimmten Funktionalitäten, welche die Simulation zur Verfügung stellt. Die Logik der meisten Anwendungen ist sehr stark mit dem zu Grunde liegendem Simulationssystem verbunden. Dies motiviert die Verwendung eines eigenständigen Moduls, das als zentrale Wissensbasis agiert. Es dient zur Datenhaltung aber auch als Koordinator für das Sammeln der verschiedenen Informationen. Somit wird die Verbindung der Anwendungslogik zur Implementierung sehr stark gelockert und eine Portierung vereinfacht. Außerdem ist die stark verzahnte Modellierung der Logik mit dem darunter liegenden Framework oft umständlich und wird bei größeren Anwendungen schnell unübersichtlich, sodass es für neue und unerfahrene Entwickler sehr schwierig ist, sich zurecht zu finden.

Hervorgehend aus den hier genannten Informationen ist die Motivation, weniger Anwendungen neu entwickeln zu müssen, sobald man sich, egal aus welchen Gründen, für den Einsatz eines anderen Frameworks entscheidet. Die Motivation der vorliegenden Arbeit leitet sich also aus dem Ziel ab, eine **längere Persistenz**[4] **der Anwendungen** zu gewährleisten. Wie in Abbildung 1.1 dargestellt ist, tragen die folgenden zwei Punkte dazu bei, dieses Ziel zu erreichen:

1. **Abstraktion:** Bessere und einfachere Möglichkeiten für die Modellierung der Anwendungslogik.
2. **Modularität:** Austauschbarkeit von Simulationskomponenten.

Resultierend aus der erläuterten Motivation ergibt sich die im folgenden Abschnitt dargelegte Zielsetzung.

[4] **Persistenz:** von lat. *persistere* „verharren“. Etwas mit dauerhafter Beschaffenheit oder Beharrlichkeit, das langfristige Fortbestehen einer Sache.

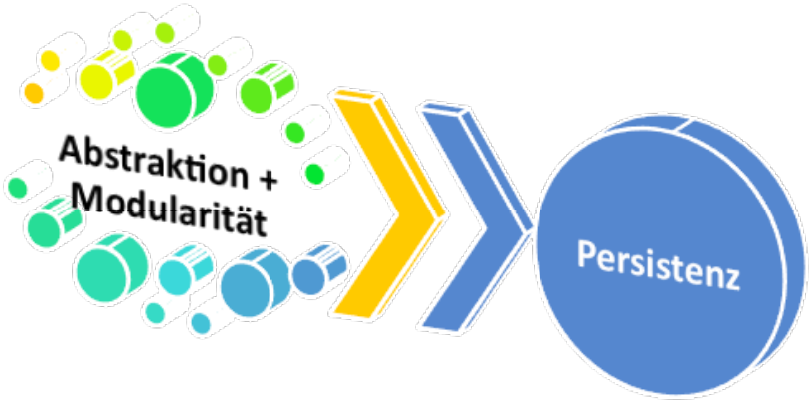


Abb. 1.1: Eine längere Persistenz wird durch einen höheren Grad der Abstraktion sowie einer möglichst modularen Struktur des Systems erreicht.

1.2 Zielsetzung und Fokus der Arbeit

Das Ziel der vorliegenden Arbeit ist es, ein System zu realisieren, welches durch seine Architektur und die Gestaltung seiner Komponenten so aufgebaut ist, dass es die gewünschte Modularität und Abstraktion umsetzen kann. Um diesen Punkten zu begegnen, konzentriert sich die vorliegende Arbeit auf folgende Ansätze:

Eine modulare Systemarchitektur:

Um den Anforderungen der **Modularität** im Hinblick auf die längere Persistenz gerecht zu werden, soll sich die Architektur des entwickelten Frameworks um ein zentrales Modul aufbauen. Diese Komponente bildet die Logik der entwickelten Anwendung ab und vereint alle Daten der zwar unabhängigen aber kollaborierenden Bestandteile. Dieses Modul ist losgelöst von der Implementierung und dem Konzept der angeschlossenen Komponenten, um es möglichst unabhängig von diesen zu halten. Somit können die spezialisierten Elemente mit geringem Aufwand ausgetauscht werden, was in einer größeren Flexibilität des Systems sowie der Anwendungen resultiert.

Um den Endanwender mit einem höchstmöglichen Spektrum von Funktionalitäten auszurüsten, ist darüber hinaus auch die Integration erprobter Methoden ein möglicher Schritt. Zu diesem Zweck werden bewährte Prinzipien aus bestehenden *Virtual Reality Frameworks* in das erstellte Simulationssystem übertragen. Zum Beispiel soll die Möglichkeit bestehen, zur Laufzeit der Simulation mittels einer oder mehrerer Skriptsprachen Einfluss zu nehmen. Dadurch wird ein *Rapid Prototyping*[5] realisiert. Die Manipulation durch Skripte soll in der vorliegenden Arbeit durch ein eigenständiges Modul ermöglicht werden, welches ausschließlich eine zentrale Wissensbasis beeinflusst.

Abstraktion der Simulationsinhalte und Anwendungslogik:

Um eine höhere Lebensdauer von Anwendungen zu gewährleisten, wird eine möglichst hohe Abstraktion der Simulationsinhalte und Anwendungslogik angestrebt. Die Modellierung soll unabhängig von speziellen Implementierungen gemacht werden. Zu diesem Zweck werden Methoden aus dem Gebiet der Künstlichen Intelligenz in das zu erstellende Framework integriert. Diese Methoden beschäftigen sich mit einer umfassenden **semantischen Modellierung virtueller Umgebungen**.

Abschließend muss abgegrenzt werden, dass es ist nicht der Fokus der Arbeit ist, neue Module für spezielle Aufgaben, wie etwa die grafische oder auditive Ausgabe, zu entwickeln. Vielmehr soll die Möglichkeit geschaffen werden, eben solche Komponenten mit geringem Aufwand in die entwickelte Architektur integrieren zu können. Zusammenfassend sollen (das Ziel eine längere Persistenz von VR Anwendungen verfolgend) die folgenden Punkte umgesetzt werden:

[5] **Rapid Prototyping** (übersetzt etwa „schneller Modellbau“): Ermöglicht die einfache Erstellung musterhafter Anwendungen.

1. **Entwicklung einer modularen Simulationsarchitektur:**

   (a) Verwendung von austauschbaren Komponenten, die ihre Daten in einen konsistenten Weltzustand integrieren

   (b) Integration von bewährten Methoden aus dem Bereich der VR-Entwicklung, um den Anwender mit vertrauten Funktionen auszustatten

2. **Semantische Modellierung virtueller Umgebungen**

   (a) Abstraktion der Simulationsinhalte und Anwendungslogik auf einer semantischen Ebene, um die Implementierung unabhängig von den verwendeten Modulen zu gestalten. Die Inhalte spiegeln die für den Benutzer wahrnehmbaren Elemente wieder, während die Anwendungslogik den Bereich beschreibt, der die internen Abläufe der Simulation regelt. Dabei spielen zum Beispiel zeitliche Abfolgen von Funktionen oder das Vorhalten eines einheitlichen Weltzustandes eine Rolle.

1.3 Forschungskontext in der Arbeitsgruppe Wissensbasierte Systeme

In der Arbeitsgruppe Wissensbasierte Systeme (kurz AG WBS) an der Technischen Fakultät der Universität Bielefeld, unter der Leitung von Prof. Dr. Ipke Wachsmuth, wird seit ca. 20 Jahren die Kombination von Computergrafik und Künstlicher Intelligenz erforscht [WBS 2014]. Die AG WBS verwendet die Möglichkeiten der Virtuellen Realität, um in verschiedenen Szenarien und Projekten zu forschen. Die Spanne der durchgeführten Projekte reicht von der virtuellen Konstruktion von Fahrzeugen über die Kommunikation mit emotionalen virtuellen Agenten zur Verbesserung der Mensch-Maschine-Interaktion bis zu der Erforschung von Kommunikation mittels Sprache und Gestik zwischen menschlichen Gesprächspartnern.

Viele Projekte haben gezeigt, dass die Integration von KI-Methoden in Simulationen der Virtuellen Realität viel versprechende Möglichkeiten bietet. Dieser damals relativ neue und bislang wenig erforschte Ansatz, wird weitergehend untersucht und auch mit dieser Dissertation weiter ausgebaut.

Virtuelle Konstruktion: In dem Projekt VIENA [Wachsmuth u. a. 1995], das aus dem Forschungsbereich „Künstliche Intelligenz und Computergrafik“ [Wachsmuth 1994] hervorging, ist es am Beispiel der Innenarchitektur erstmals gelungen ist, eine intelligente Mensch-Maschine-Kommunikation mit einem technischen System für die Handhabung computergrafischer Entwurfsumgebungen zu verwirklichen. Darauf aufbauend wurde die virtuelle Konstruktion in weiteren Projekten erforscht. Im CODY [Jung u. Wachsmuth 1995] *Virtual Constructor* wurden verschiedene Wissensbasen vorgehalten, um geometrische Informationen und logische Repräsentationen zu verarbeiten. Die Beschreibungen wurden dabei mit einem geometrischen Objekt verknüpft, sodass beim Zusammenbau virtueller Baugruppen logische Beziehungen der einzelnen Objekte zueinander geschlossen werden konnten. Im Kontext des von der Deutschen Forschungsgesellschaft geförderten Projektes **Virtuelle Werkstatt** wurde die virtuelle Konstruktion mittels VR-Technologie, kombiniert mit möglichst natürlicher Interaktion mit Sprache und Gestik, erforscht. Dieses Projekt verbindet dabei auch, als eines der ersten in voll immersiver Virtueller Realität, die Disziplinen der Computergrafik und Künstlichen Intelligenz. Ein umfassender Überblick über das Projekt wurde in [Fröhlich u. a. 2009b] veröffentlicht. Die virtuelle Konstruktion wird in Kapitel 6 als ein Anwendungsbeispiel wieder aufgegriffen.

Mensch-Maschine-Kommunikation: Ein Beispiel für die Forschung im Bereich der Mensch-Maschine-Kommunikation ist der virtuelle Agent MAX [Kopp u. a. 2003]. Ursprünglich im Rahmen einer Dissertation entstanden, wurde er in vielen Projekten weiterentwickelt, unter anderem im Sonderforschungsbereich 360 – „Situierte Künstliche Kommunikatoren“. MAX nimmt in der virtuellen Realität menschliche Gestalt

an und verkörpert eine neuartige Schnittstelle der Mensch-Maschine-Interaktion. Es wird das Ziel verfolgt, mit ihm auf natürlichsprachliche Art und Weise zu kommunizieren und ihn mit kommunikativer Intelligenz auszustatten.

Mensch-Mensch-Kommunikation: Mittels verschiedener Szenarien in der virtuellen Realität wird auch die zwischenmenschliche Kommunikation weiter ergründet. Dabei wird besonderer Wert auf die Kommunikation mit Sprache und Gestik sowie die *kollaborative*[6] Interaktion menschlicher Partner in der virtuellen Realität gelegt. Unter anderem werden Projekte im Kontext des Sonderforschungsbereiches 673 – „Alignment in Communication“ durchgeführt [Pfeiffer-Lessmann u. Wachsmuth 2008]. Das von der Europäischen Union geförderte Projekt PASION [Pfeiffer u. Wachsmuth 2008] beschäftigte sich außerdem mit der sozialen Kommunikation über Netzwerke.

Einige der im Laufe der Jahre entstandenen VR-Anwendungen sind heute nur noch schwer zu warten und weiter zu betreiben, da in der Vergangenheit einige Architektur- und Softwarewechsel vorgenommen wurden. Dieses ist ein natürlicher Prozess.

Da der Autor dieser Arbeit an vielen dieser Projekte direkt oder indirekt beteiligt war, ist es auch eine persönliche Motivation, Möglichkeiten zu schaffen, die es erlauben, vergangene Projekte weiter betreiben und warten zu können. Dabei wird versucht, diese persönlichen Erfahrungen mit den durchgeführten Projekten und dem Umgang der eingesetzten Frameworks einzubeziehen.

1.4 Aufbau der vorliegenden Arbeit

Die vorliegende Arbeit ist in insgesamt sieben Kapitel unterteilt. Während in **Kapitel 1** in das Thema eingeführt wurde sowie die Motivation und Ziel-

[6] **Kollaboration:** von lat. *co-* = „mit-“, *laborare* = „arbeiten“)

setzung dargestellt wurden, werden in **Kapitel 2** verschiedene Ansätze zur Entwicklung von Simulationssystemen vorgestellt. Einzelne Aspekte relevanter Arbeiten werden herausgearbeitet und in Relation zu der vorliegenden Arbeit gesetzt. Dabei werden zwei grundlegend unterschiedliche Konzepte unterschieden sowie deren Vor- und Nachteile analysiert.

Kapitel 3 stellt die Hintergründe der semantischen Modellierung virtueller Umgebungen vor. Diese geben einen tieferen Einblick in die verwendeten Konzepte und Methoden. Hierbei werden sowohl abstrakte wie konkrete Konzepte, auf denen die erarbeitete Architektur und deren Implementierung beruhen, Punkt für Punkt vorgestellt.

In **Kapitel 4** wird das Konzept der erarbeiteten Architektur vorgestellt. Im Besonderen wird gezeigt, wie die in Kapitel 3 erläuterten Aspekte zu der Umsetzung beitragen und wie diese in die Domäne der Softwarearchitekturen für Virtuelle Realität integriert wurden. Dabei werden Anforderungen definiert, ein Konzept entwickelt und die exemplarische Integration der verschiedenen Bestandteile dargelegt.

Die Implementierung wird in **Kapitel 5** vorgestellt. Dabei wird gezeigt, wie die theoretischen Methoden und die erarbeiteten Konzepte in ein umgesetztes Softwareframework einfließen und die semantische Modellierung konkret realisiert wird.

Kapitel 6 zeigt die virtuelle Konstruktion als Beispielanwendung für das entwickelte Framework und geht auf die Vorteile ein, die auf Grund seiner Architektur in die Anwendungen einfließen.

Den Abschluss der Arbeit bildet eine zusammenfassende Betrachtung in **Kapitel 7**. Im Zuge dieser wird ein Fazit über die erreichten Ergebnisse gegeben sowie eine kritische Analyse über mögliche Verbesserungen und weitere Folgearbeiten durchgeführt.

1.5 Typographische Konventionen

Um es dem Leser zu erleichtern dem Verlauf der Arbeit besser folgen zu können, werden an dieser Stelle einige typographische Konventionen festgelegt. Im Folgenden werden bestimmte Eigennamen, beispielsweise die Namen spezieller Softwareframeworks, in KAPITÄLCHEN gesetzt. Fremdwörter und fachspezifische Bezeichnungen werden bei ihrem ersten Vorkommen *kursiv* gedruckt. Hervorzuhebende Begriffe werden durch ein **fett gedrucktes** Layout gekennzeichnet. Verwendete Klassen- und Funktionsnamen sowie sich auf den Quelltext beziehende Begriffe werden in `Schreibmaschinenschrift` dargestellt. Begriffe, die einer weiteren kurzen Erklärung bedürfen, werden in einer Fußnote[7] näher ausgeführt.

[7] **Fußnote:** Eine Anmerkung, die aus dem Fließtext ausgelagert wird, um den Text flüssig lesbar zu gestalten

KAPITEL

2

SIMULATIONSSYSTEME FÜR VIRTUELLE UMGEBUNGEN

Zur Entwicklung multimodaler virtueller Umgebungen sind verschiedene Frameworks implementiert worden. Die damit in Verbindung stehenden Paradigmen und Lösungsansätze spannen den inhaltlichen Rahmen des folgenden Kapitels auf. Dabei werden die Vor- und Nachteile verschiedener ausgewählter Vorgehensweisen in den Kontext der vorliegenden Dissertation gebracht. Außerdem werden einige Bereiche aus dem Kontext der Entwicklung virtueller Welten vorgestellt, welche in Verbindung zur hier erstellten Arbeit stehen. Zunächst wird kurz die Virtuelle Realität als Anwendungsdomäne erläutert, wobei ein Überblick über das Themengebiet gegeben wird.

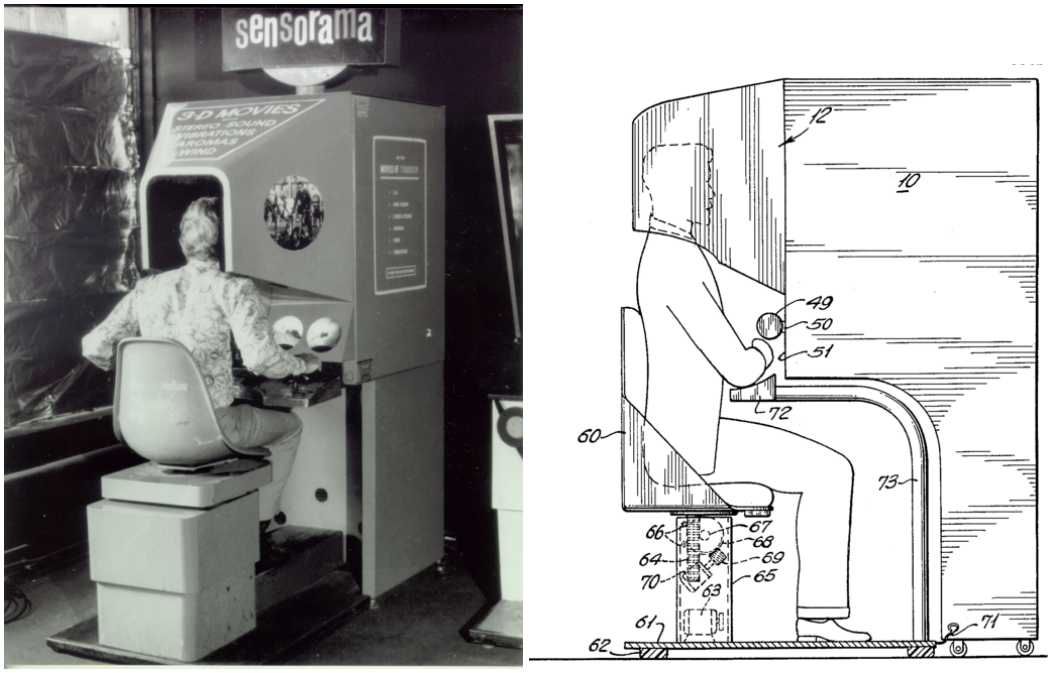
Der Begriff „Virtuelle Realität“ hat seinen Ursprung in den 1980er Jahren und erschien 1987 erstmals im „Oxford English Dictionary“. Es existieren viele verschiedene Definitionen, sodass keine allgemein gültige angegeben werden kann. Wahlweise werden entweder technische Aspekte in den Mittelpunkt gerückt, oder die Definition wird um den Benutzer herum aufgebaut, der die virtuelle Welt erleben soll. In den folgenden beiden Definitionen werden Beispiele dieser zwei Arten gegeben:

Definition 1 (Virtuelle Realität)**.** *[...] eine mittels Computer simulierte Wirklichkeit oder künstliche Welt, in die Personen mithilfe technischer Geräte sowie umfangreicher Software versetzt und interaktiv eingebunden werden [Brockhaus 1998].*

Definition 2 (Virtuelle Realität)**.** *[...] bezeichnet ein neuartiges Kommunikationsmedium, das die unmittelbare Wechselwirkung des Menschen mit rechnergenerierten Darstellungen konkreter oder abstrakter Ereignisse und Sachverhalte erlaubt [Wachsmuth 1998].*

Während sich Definition 1, ähnlich der vorliegenden Arbeit, auf die technischen Grundlagen konzentriert, rückt Definition 2 die Wechselwirkung des Benutzers mit der computergenerierten Welt in den Vordergrund. Die Verbesserung der Benutzererfahrung steht beispielsweise bei [Fröhlich u. Wachsmuth 2013] im Fokus.

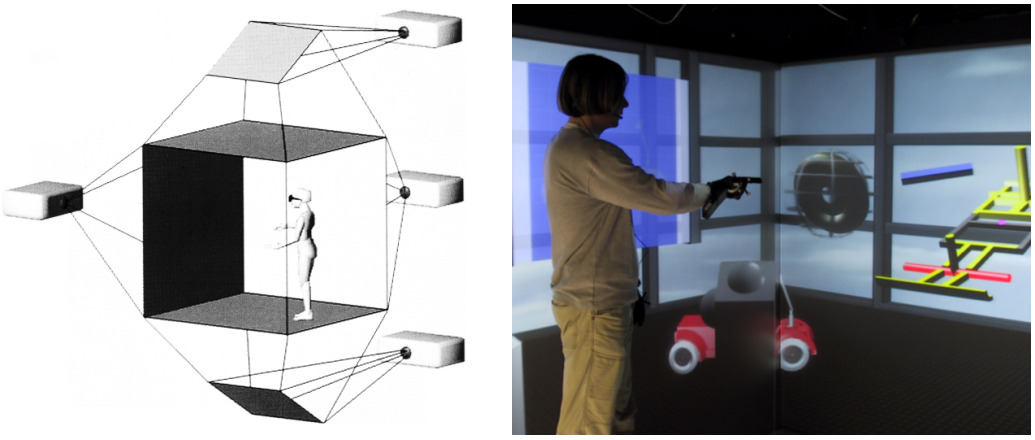
Obwohl nicht eindeutig geklärt ist, wer den Begriff originär eingeführt hat, wird in diesem Zuge oft der 1982 erschienene Science-Fiction Roman „The Judas Mandala“ von Damien Broderick genannt [Broderick 1982]. Als einer der Pioniere, die die Entwicklung heutiger VR Systeme vorangetrieben haben, ist Jaron Lanier zu nennen [Lanier 1992]. Technische Systeme, die in eine ähnliche Richtung gehen wie aktuelle VR-Installationen, existierten allerdings schon weit vorher. Das Sensorama von Morton Heilig wurde bereits 1957 entwickelt und 1962 unter Patent gestellt [Heilig 1962]. Es war in der Lage multimodale Eindrücke darzustellen, darunter stereoskopische Bilder, akustische Ausgaben und Feedback durch Vibrationen. Für eine Virtuelle Realität, wie sie in heutigen Systemen dargestellt wird, fehlte dem Sensorama allerdings die Interaktion des Benutzers, der sich die Welt als passiver Zuschauer ansehen konnte. Abbildung 2.1 zeigt einen Benutzer während er einen „Film“ im Sensorama anschaut (a) sowie eine schematische Illustration des Sensoramas aus Heiligs Patent (b).



(a) (b)

Abb. 2.1: (a) Das Sensorama in Aktion [Heilig 2014] und (b) eine Illustration aus Morton Heiligs Patent [Heilig 1962]

Während also schon früh Systeme existierten, die sich einer Virtuellen Realität annäherten, machte die Entwicklung von modernen VR-Systemen in den 1990er Jahren einen großen Sprung. Die Vision einer virtuellen Realität war in dieser Zeit häufig in Science Fiction Medien, wie etwa das sogenannte Holodeck in Star Trek, präsent und erhöhte so die Wahrnehmung des Themas in der Öffentlichkeit. Mit der Erfindung des Cave Automatic Virtual Environment (kurz CAVE) gelang es Cruz-Neira, Sandin, DeFanti, Kenyon, u. Hart [1992] ein System zu entwickeln, das heute in vielen VR-Anwendungen verwendet wird. Eine CAVE stellt ein immersives VR-System dar, das aus mehreren Projektionsflächen besteht, die stereoskopische Bilder anzeigen. Der Benutzer hat nach dem Betreten das Gefühl, dass sich die virtuelle Welt um ihn herum aufbaut. Abbildung 2.2 illustriert die schematische Abbildung einer CAVE (a) und eine exemplarische Anwendung (b).



(a) (b)

Abb. 2.2: (a) Schematische Darstellung einer CAVE [Cruz-Neira u. a. 1992] und (b) eine exemplarische Anwendung innerhalb einer CAVE Umgebung

Für die Simulation einer virtuellen Umgebung braucht es neben einem Hardware-System auch noch spezialisierte Software. Diese muss in der Lage sein, die verschiedenen Eindrücke zu generieren sowie die Interaktionen der Benutzer zu verarbeiten. Auf dem Gebiet der Virtuellen Realität existiert eine Vielzahl von Frameworks mit verschiedenen grundlegenden Konzepten. Dabei werden im Wesentlichen zwei verschiedene Ansätze verwendet:

1. **Monolithisch** (von griechisch *monolithos* „Ein Stein“):

   Ein monolithischer Ansatz zeichnet sich im Allgemeinen dadurch aus, dass er als ganzheitlicher Block angesehen wird. Im vorliegenden Kontext sind somit die Bestandteile des Simulationssystems fest miteinander verbunden und nicht zu trennen. In einer monolithischen Architektur herrschen in der Regel eine strikte Hierarchie der Einzelteile und eine streng festgelegte Kommunikationsstruktur. Eine solche Architektur ist in Abbildung 2.3 abgebildet.

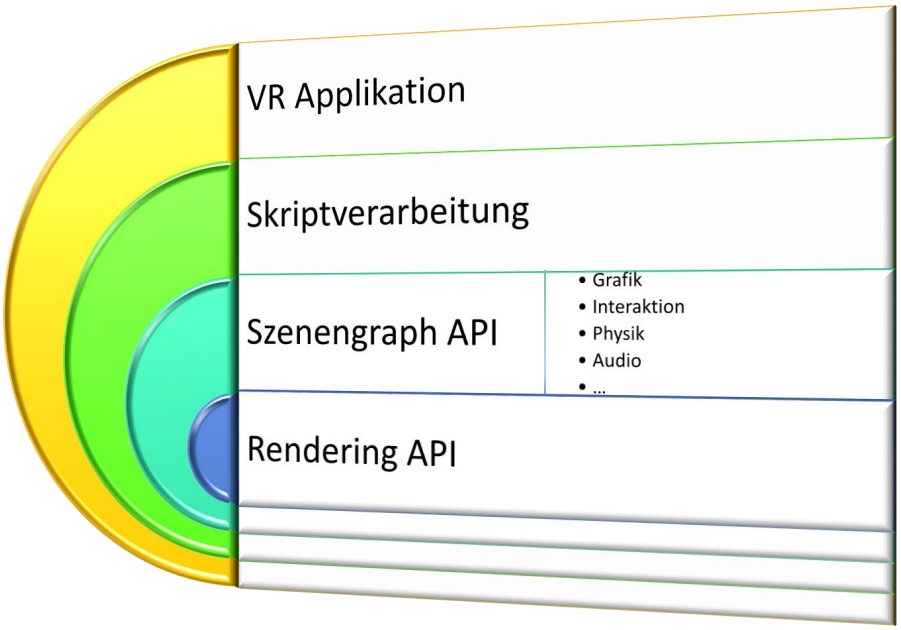


Abb. 2.3: Illustration einer monolithischen Simulationsarchitektur. Die Bestandteile unterliegen einer strengen hierarchischen Struktur. Zusätzliche Modalitäten werden oft in den Szenengraphen integriert.

2. **Modular** (von lateinisch *modulus* "Maß"):

   Module können in der Softwareentwicklung als eine funktionsorientierte Gruppe von Befehlen betrachtet werden. Eine modulare Architektur zeichnet sich dadurch aus, dass ihre Einzelteile (Module) größtenteils unabhängig von einander agieren und dabei über ein definiertes Protokoll miteinander kommunizieren. Abbildung 2.4 illustriert eine modulare Architektur.

Viele der in der Vergangenheit entwickelten Frameworks verwenden einen monolithischen Ansatz, bei dem verschiedene Funktionalitäten in die Grafikkomponente integriert, beziehungsweise um die Grafik herum entwickelt wurden. Resultierend hieraus sind einzelne Funktionen nur schwer (wenn überhaupt) austauschbar, sodass im Falle eines Wechsels des Frameworks alle Bereiche einer Anwendung neu implementiert werden müssen.

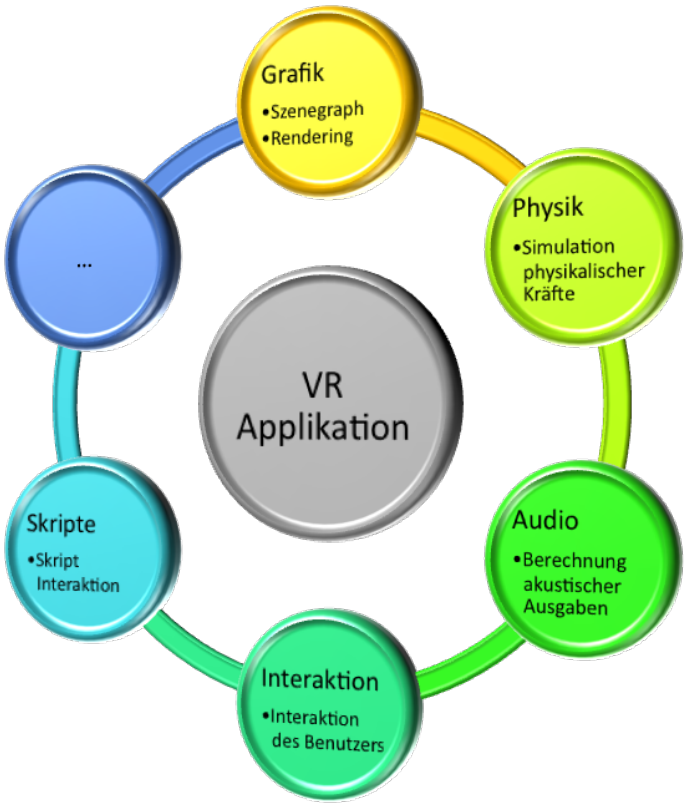


Abb. 2.4: Illustration einer modularen Simulationsarchitektur. Die Module agieren unabhängig voneinander und unterliegen keiner hierarchischen Struktur.

Einige Systeme verfolgen einen modularen Ansatz, bei dem einzelne Komponenten getauscht werden können, um somit die Portierung auf alternative Systeme zu erleichtern. Die einzelnen Module sind stark funktionsorientiert und übernehmen dedizierte Aufgaben. Die Kommunikation reicht von einem schlanken Austausch der wichtigsten Daten bis hin zu einem kompletten Abgleich des existierenden Zustands der simulierten virtuellen Umgebung (Weltzustand).

Im weiteren Verlauf werden die beiden beschriebenen Ansätze anhand einiger Frameworks beispielhaft diskutiert und näher ausgeführt.

2.1 Monolithische Ansätze

In den folgenden Abschnitten werden stellvertretend für den monolithischen Ansatz die vier bekannten monolithischen Frameworks Avango, OpenSG, OpenSceneGraph und InstantReality vorgestellt.

2.1.1 Avango

Die Entwicklung des Avango-Frameworks [Tramberend 1999] begann in den späten 1990er Jahren unter dem damaligen Namen AVOCADO. Zur graphischen Darstellung verwendet es OpenGL[1] Performer™ von der Firma SGI. Aufbauend auf diesem Framework wurden weitere Schichten entwickelt, welche eine Vielzahl von erweiterten Funktionalitäten bereit stellen. Die Avango Architektur ist in Abbildung 2.5 dargestellt.

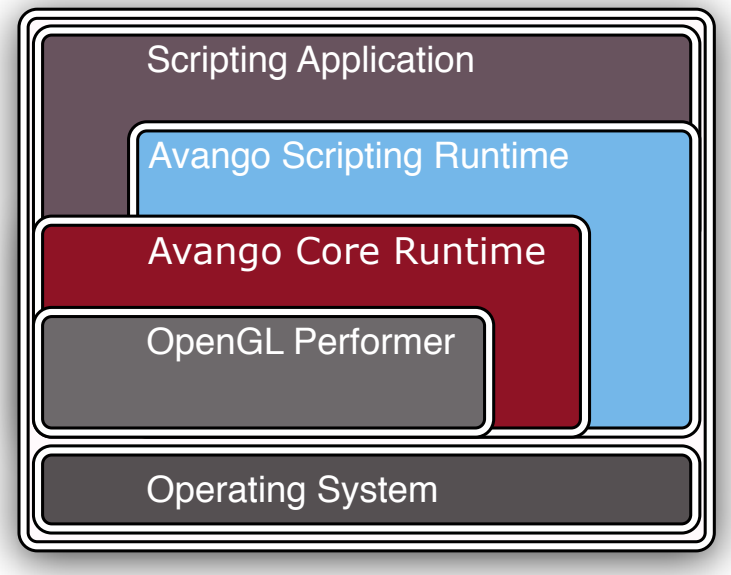


Abb. 2.5: Übersicht der verschiedenen Schichten der Avango Architektur.

[1] **OpenGL:** ***O****pen* ***G****raphics* ***L****ibrary*. Eine Spezifikation für eine Programmierschnittstelle zur Entwicklung computergrafisch dargestellter Inhalte

Von der untersten bis zur höchsten sind folgende Schichten abgebildet:

1. Das Betriebssystem (Linux).
2. Das für die grafische Ausgabe zuständige *Rendering*-System (OpenGL Performer™).
3. Der Avango-Kern ist eine in C++ implementierte Schicht. Er erweitert die darunter liegenden Szenengraphknoten um einen Feldmechanismus und die damit verbundenen Funktionen, wie zum Beispiel das Propagieren von Werten von einem Knoten zum Nächsten durch Verbindung der entsprechenden Felder.
4. Ein Skriptinterpreter (Elk Scheme), um die Anwendungsentwicklung interaktiver zu gestalten.
5. Die geskriptete Applikation.

Der Fokus von Avango liegt auf der grafischen Ausgabe der simulierten Szene. Die verwendeten Szenenknoten sind allerdings mehr als nur grafische Objekte eines herkömmlichen Szenengraphen, wie er vom darunter liegenden Renderer verwendet wird. Vielmehr sind Avango-Knoten aktive Elemente, die ihre Zustände in Feldern speichern und diese mit anderen Knoten der Szene verbinden können. Dieser Mechanismus bietet eine praktikable Möglichkeit dynamische und interaktive Szenen zu definieren. Einzelne Elemente können über ihre Verbindungen auf Änderungen anderer Knoten reagieren. Avango selbst ist über die Knoten, die Zustände sowie die bestehenden Verbindungen informiert und kann für ein konsistentes Verhalten der gesamten Szene sorgen.

Um ein Rapid Prototyping der Simulation zu ermöglichen, wird die Skriptschnittstelle verwendet. Durch ihre Verwendung ist es möglich, zur Laufzeit Änderungen an der Szene oder der Anwendungslogik vorzunehmen. Eine typische Avango-Applikation kann in der Regel komplett in der Skriptsprache definiert werden, indem die bestehenden C++ Knoten instanziiert und miteinander verbunden werden. Die Verbindung verschiedener Knoten definiert

hierbei einen großen Teil der Anwendungslogik. Das heißt im Gegenzug, dass kein selbstständiges Modul für die Definition dieser Logik existiert. Die Wissensrepräsentation der Applikation ist in das Grafiksystem integriert worden.

Da eine typische Anwendung in der virtuellen Realität nicht mit einer Interaktion über Maus und Tastatur auskommt, bietet das Framework außerdem einfache Möglichkeiten, weitere Interaktionstechnologien einzubinden. Dazu sind bereits eine Reihe gängiger Interaktionsgeräte vordefiniert und für den Einsatz bereit. Ebenso lassen sich auch verschiedene Displayvarianten durch den Einsatz simpler Methoden ansteuern.

Zusammenfassend lässt sich sagen, dass die AVANGO Architektur über die eines Szenengraphen hinausgeht. Sie basiert auf einer Rendering-Schnittstelle und erweitert diese um integrierte konnektive Felddefinitionen. Diese Metapher bringt jedoch auch einige Nachteile mit sich, wenn es um die Übersichtlichkeit und vor allem um die Wiederverwertbarkeit geht. Des Weiteren fehlt eine komfortable Anbindung anderer Komponenten, zum Beispiel akustische Ausgaben, welche für eine komplette VR-Simulation von Bedeutung sind.

Seit dem Jahr 2008 existiert eine überarbeitete Version von AVANGO. Das sogenannte AVANGO NG oder AVANGO 2.0 [Kuck u. a. 2008] hat einige Änderungen durchlaufen, wobei die generelle Idee der Architektur jedoch weitestgehend unverändert geblieben ist.

2.1.2 OpenSG

Die Idee zur Entwicklung von OPENSG entstand bereits im Jahr 1999 – mit der Entwicklung wurde Anfang der 2000er Jahre beim Fraunhofer-Institut für grafische Datenverarbeitung (kurz: IGD) begonnen [Reiners u. a. 2002]. Ebenso wie AVANGO basiert auch OPENSG auf der Szenengraph-Metapher, allerdings setzt OPENSG dabei auf eine eigene Implementierung, die einige Vorteile gegenüber anderen verfügbaren Systemen bietet. Drei der signifikantesten Fortschritte liegen in den folgenden Bereichen:

1. **Erweiterbarkeit**

   OpenSG bietet einfache Möglichkeiten, über eine Schnittstelle neue Szenengraphknoten selber zu definieren, mittels derer eigene Funktionlitäten implementiert werden können. Eine gute Erweiterbarkeit ist eine Voraussetzung für ein funktionales Simulationssystem, da immer wieder spezielle Funktionalitäten benötigt werden, die das zu Grunde liegende System nicht von Hause aus mit sich bringt.

2. **Multithread-Sicherheit**

   Die Vorteile einer *Multithread*[2]-Funktionalität liegen in der Performanz des Simulationssystems. Zieht man in Betracht, dass in einer solchen monolithischen Architektur nahezu die komplette Datenhaltung sowie die Anwendungslogik von der Szenengraph-Struktur verwaltet wird, gewinnt eine Multithread-Sicherheit umso mehr an Bedeutung. In der heutigen Zeit wird mehr und mehr Wert auf multiple Prozessorkerne gelegt. Um die volle Kapazität eines aktuellen Computers nutzen zu können, ist es unabdingbar seine verschiedenen Kerne zu verwenden.

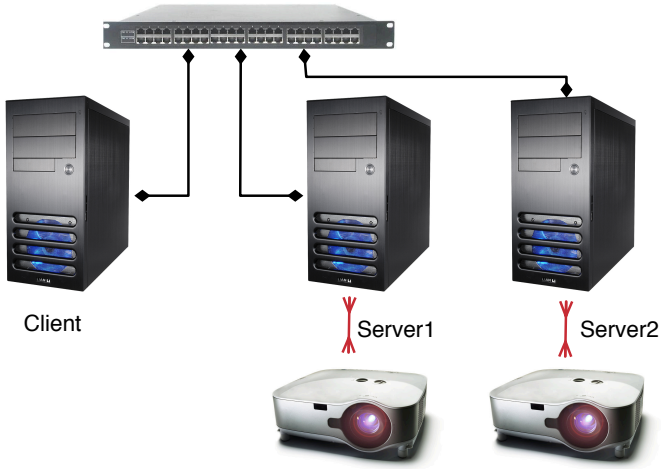
3. **Clustering**

   Der Begriff des *Clusterings*[3] bezieht sich in dieser Domäne auf die Fähigkeit die Last der Applikation auf unterschiedliche vernetzte Computer zu verteilen. Dies ist immer dann von Nöten, wenn die Rechenleistung eines einzelnen PCs nicht mehr ausreicht, um die Simulation flüssig darzustellen. Typische Anwendungsfälle sind Multidisplay-Setups, in denen es die Regel ist, dass ein Rechner alleine die nötige Last nicht in vollem Umfang bewältigen kann. Hier arbeitet OpenSG mit einem sogenannten verteilten Szenengraphen. Das heißt, es läuft auf jedem beteiligten Rechner ein Client. Zwischen diesen werden dann nur Änderungen des Szenengraphen propagiert. Dieses Prinzip verursacht mehr Last auf den einzelnen Rechnern, ist allerdings sehr sparsam in Bezug

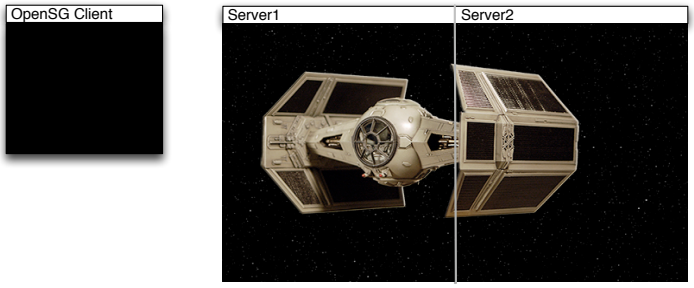
[2] **Multithread:** auch Nebenläufigkeit. Bezeichnet das gleichzeitige Abarbeiten mehrerer Ausführungsstränge innerhalb eines einzelnen Prozesses.

[3] **Clustering:** ein Cluster (von engl. „Gruppe“ oder „Haufen“) bezeichnet eine Anzahl von vernetzten Computern. Durch die Vernetzung kann die Rechenleistung erhöht werden.

auf den Datendurchsatz im Netzwerk. Zu synchronisierende Bereiche der Applikation müssen im Sourcecode gekennzeichnet werden, damit sie in den relevanten *Changelists*[4] registriert werden können. Abbildung 2.6 zeigt (a) eine typische Cluster-Anwendung und (b) das dadurch erzeugte Resultat.



(a)



(b)

Abb. 2.6: (a) Ein typisches Cluster-Setup mit (b) dazugehöriger Ausgabe. Illustrationen nachempfunden von http://www.opensg.org

[4] **Changelist:** (von engl. „verändern“) speichert Änderungen in einem Szenengraphen, um diese später zu verarbeiten

OpenSG ist also ein moderner, effizient arbeitender und leicht erweiterbarer Szenengraph. Allerdings fehlt, ähnlich wie bei Avango, die Möglichkeit auf einfache Art und Weise weitere Modalitäten zu integrieren. Nichtsdestotrotz ist es möglich, beispielsweise Audioknoten zu entwickeln, allerdings mit erheblichem Aufwand. Die selbst entwickelten Knoten sind wiederum im Szenengraphen verankert und somit sehr eng an das Grafiksystem gekoppelt. OpenSG regelt die komplette Anwendungslogik in dem Szenengraph. Damit wird die Wiederverwendung einer in OpenSG geschriebenen Applikation in einem anderen System in der Praxis nahezu unmöglich.

2.1.3 OpenSceneGraph

Die Entwicklung des OpenSceneGraph-Frameworks wurde 1998 von Don Burns als Hobby gestartet [Burns u. Osfield 2004]. Das Framework orientiert sich sehr stark an OpenGL Performer™ und basiert auf den selben Prinzipien. Allerdings hatte es den Vorteil, dass die Implementierung bereits Ende 1999 auf *open source* Basis gestaltet wurde und somit sehr viel zugänglicher war. Die Entwicklung begann circa zur gleichen Zeit wie die von OpenSG. Es war in seiner Anfangszeit vor allem eine gute Alternative für Entwickler mit Erfahrungen in OpenGL Performer™. Jedoch bot OpenSceneGraph mehr Möglichkeiten der Eigenentwicklung. Nachdem die Firma SGI die Weiterentwicklung des Performer Frameworks eingestellt hatte, bot OpenSceneGraph eine sinnvolle Alternative. Seitdem wurde das Framework in einer Vielzahl von kommerziellen wie auch nicht-kommerziellen Projekten eingesetzt. Die aktuelle Version 3.2 wurde 2013 veröffentlicht.

Die Abbildung 2.7 zeigt den Aufbau der OpenSceneGraph Architektur. Auf unterster Ebene sind die möglichen Betriebssysteme beispielhaft dargestellt. Darüber liegt zum Anzeigen der Szene die Grafikschnittstelle OpenGL. In der Mitte der Abbildung ist der Kern des OpenSceneGraph *Middleware*[5] Frameworks zu sehen, der durch *Plugins*[6] erweitert

[5] **Middleware:** (von engl. „Zwischenanwendung“) Bezeichnet Programme, die zwischen Anwendungen vermitteln.

[6] **Plugin:** Eine Erweiterung, die zur Laufzeit eingebunden werden kann.

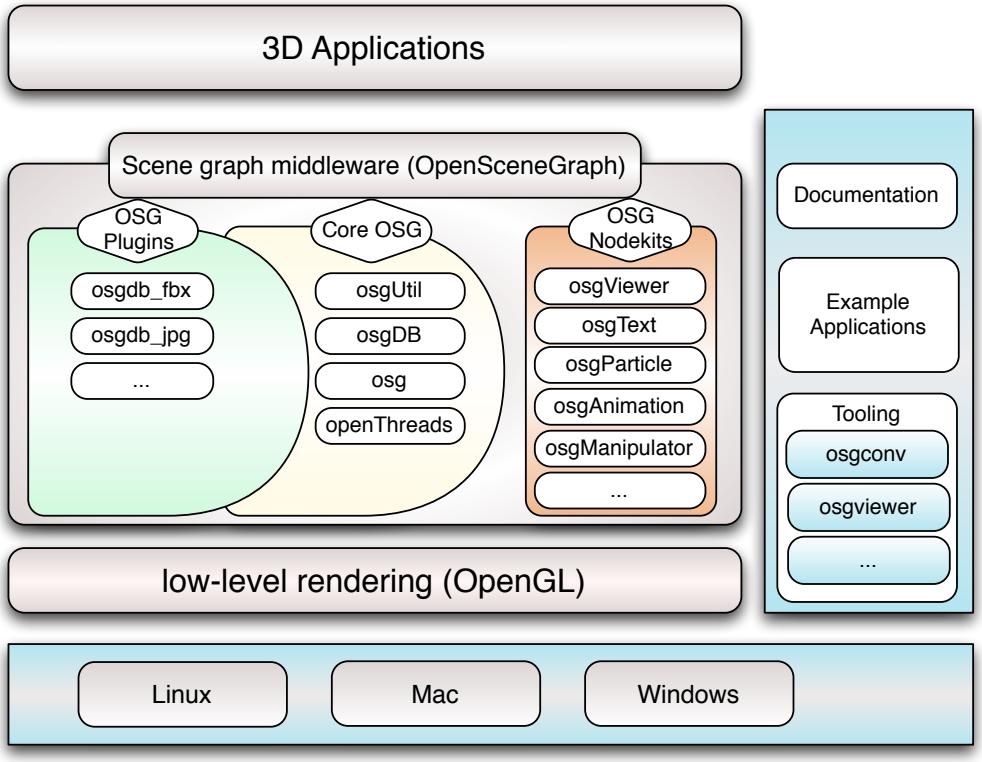


Abb. 2.7: Übersicht über die verschiedenen Schichten der OpenSceneGraph Architektur. Illustration nachempfunden von http://de.wikipedia.org/wiki/OpenSceneGraph

werden kann. Außerdem werden noch verschiedene Bibliotheken mitgeliefert, welche unterschiedliche Funktionen übernehmen, zum Beispiel ein Modul zur Animation von Objekten (`osgAnimation`). Auf OpenSceneGraph basierend werden von dem Entwickler 3D-Applikationen entwickelt. Das Framework befreit somit den Benutzer von der Low-Level OpenGL Programmierung und stellt viele nützliche Abstraktionsschichten bereit.

Das System ist komplett in C++ geschrieben und nutzt OpenGL zur Grafikgenerierung sowie eine darüber liegende klassische Struktur eines Szenengraphen. Die Stärken liegen in der Performanz, Skalierbarkeit und seiner Portabilität:

1. **Performanz**

   OpenSceneGraph unterstützt verschiedene Optimierungsverfahren, zur Behandlung des Sichtbarkeitsproblems in der Computergrafik[7]. Darüber hinaus werden auch Teile des OpenGL- Standards unterstützt. Im Gesamten betrachtet, machen diese Eigenschaften OpenSceneGraph zu einem der performantesten Szenengraph-Frameworks.

2. **Skalierbarkeit**

   OpenSceneGraph skaliert sehr gut mit unterschiedlichen Systemstrukturen. Dazu gehören Architekturen mit multiplen Prozessoren oder Grafikkarten sowie Cluster-Systeme. Dies wird dadurch ermöglicht, dass der zentrale Szenengraph mehrere Kontexte für OpenGL-Display-Listen und Texturobjekte unterstützt. Die Unterstützung für multiple Grafikkontexte und Multithreading ist nativ im `osgViewer` enthalten.

3. **Portabilität**

   Der Kern des Szenengraphen ist so gestaltet, dass er nur minimale Abhängigkeiten zu dem darunter liegenden Betriebssystem beinhaltet. Er braucht nur wenig mehr als eine Standard C++ Umgebung und OpenGL. Darüber hinaus ist die zentrale Grafikbibliothek unabhängig vom darunter verwendeten Fenstersystem. Dieses macht es dem Benutzer einfach, eigene fensterspezifische Bibliotheken und Anwendungen zu integrieren. Die OpenSceneGraph Distribution enthält bereits native Unterstützung für Windows, Linux sowie Mac OSX. Der `osgViewer` kann auch in andere Fenster-Toolkits integriert werden.

Zusammenfassend lässt sich sagen, dass OpenSceneGraph einen gut zugänglichen Szenengraph darstellt, welcher effizient arbeitet und neueste OpenGL Standards nutzt. Allerdings muss man einschränkend konstatieren, dass es sich „nur“ um ein Szenengraph-System handelt. Die Anwendungslogik wird von dem Anwender komplett in C++ geschrieben und weitere Modalitäten müssen durch Bibliotheken von außen integriert werden.

[7] Zum Beispiel: View Frustum Culling, Occlusion Culling und Level of Detail (LoD).

2.1.4 InstantReality

Bei InstantReality [Fellner u. a. 2009] handelt es sich um ein *Mixed-Reality* (MR) System. Es kombiniert verschiedene Komponenten zu einer konsistenten Schnittstelle. Diese Komponenten werden von dem Fraunhofer-Institut IGD sowie im Rahmen des Zentrums für Foren in der grafischen Datenverarbeitung (kurz: ZGDV) in enger Zusammenarbeit mit industriellen Partnern entwickelt. Das Ziel ist die Entwicklung eines möglichst einfachen Anwendungsinterfaces, das auf dem neuesten Entwicklungsstand bezüglich realistischer Darstellung, 3D-Benutzerinteraktion und immersiver Display-Technologie ist.

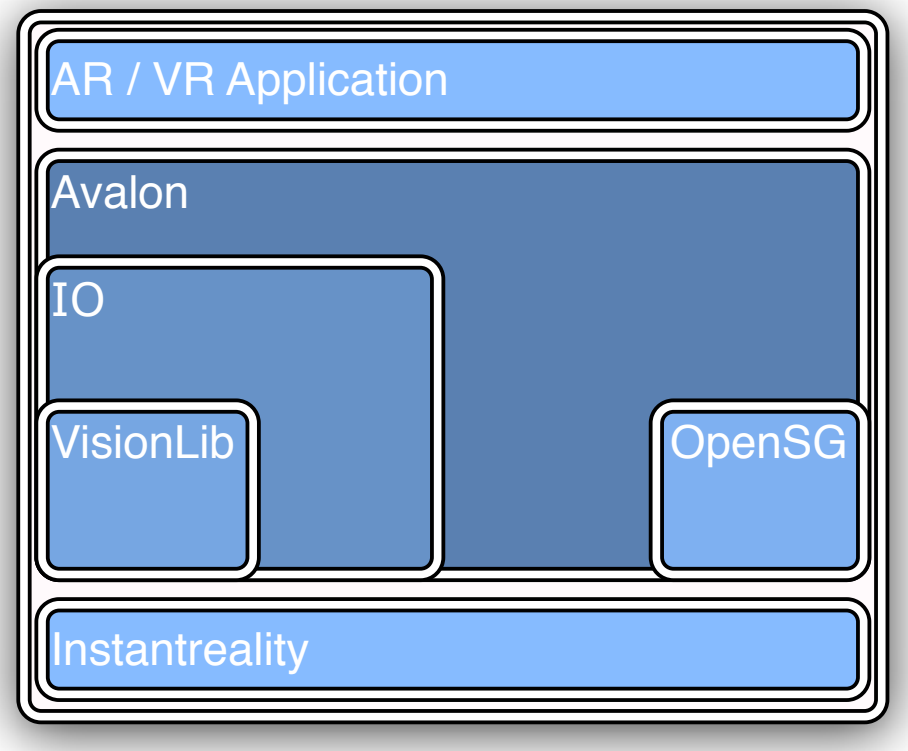


Abb. 2.8: Übersicht über die verschiedenen Schichten der InstantReality Architektur. Illustration nachempfunden von http://www.instantreality.org

Abbildung 2.8 zeigt den Aufbau der Architektur mit den verwendeten Komponenten. Es fällt auf, dass es sich hierbei nicht um ein klassisches geschichtetes System handelt, sondern eher um einen „Baukasten“, bei dem nicht alle Komponenten aufeinander basieren. Zur grafischen Darstellung wird das in Abschnitt 2.1.2 vorgestellte OpenSG eingesetzt. InstantReality unterstützt verschiedene industrielle Standards. **X3D**[8] wird als Datenformat zur Anwendungsentwicklung verwendet. Es ist die *XML*[9]-basierte Weiterentwicklung des *VRML*[10]-Formates. VRML ist seit etlichen Jahren in verschiedenen Iterationen ein beliebter Standard in der VR-Entwicklung. Listing 2.1 zeigt ein simples Beispiel, welches eine dreidimensionale Box (siehe Zeile 8) erzeugt.

```
1  <X3D>
2      <Scene>
3          <Background skyColor='1 1 1'/>
4          <Viewpoint description='Book View'
5           orientation='-0.747 -0.624 -0.231 1.05'
6           position='-1.81 3.12 2.59'/>
7          <Shape>
8              <Box size='1 2 3'/>
9              <Appearance>
10                 <Material/>
11             </Appearance>
12         </Shape>
13     </Scene>
14 </X3D>
```

Listing 2.1: Definition einer dreidimensionalen Box in X3D

Obwohl das abgebildete Beispiel nicht sehr umfangreich ist, verdeutlicht es die Vorteile XML-basierter Programmierung. Es liegt eine übersichtliche Struktur vor, die auch für nicht erfahrene Programmierer verständlich ist. Der Vorteil eines standardisierten Datenformates liegt nicht zu Letzt in der

[8] **X3D:** EXtensible 3D. Eine Beschreibungssprache für 3D Modelle.

[9] **XML:** EXtensible Markup Language. Eine Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien.

[10] **VRML:** Virtual Reality Modelling Language. Beschreibungssprache für 3D-Szenen

besseren Wiederverwertbarkeit der entwickelten Anwendung. Das Prinzip der Applikationsmodellierung mittels XML wird im weiteren Verlauf dieser Arbeit aufgegriffen und in dem entwickelten Framework verwendet. Eine in X3D entwickelte Anwendung kann abgesehen von den spezifischen Knoten des InstantReality Frameworks auch in anderen Frameworks abgespielt werden, die diesen Standard unterstützen. Resultierend hieraus muss, im Gegensatz zu den anderen bisher vorgestellten Frameworks, im Falle einer Portierung auf ein anderes System, nicht die komplette Anwendungslogik neu gestaltet werden.

2.2 Modulare Ansätze

Folgend werden nun exemplarisch die zwei modularen Ansätze der Systementwicklung FlowVR und Lightning vorgestellt. Diese arbeiten nicht immer so effizient wie monolithische Ansätze, da sie einen gewissen Mehraufwand auf Grund der Kommunikation zwischen den einzelnen Modulen beinhalten, bieten allerdings Vorteile in anderen Bereichen. Sie sind basierend auf ihrer modularen Struktur aber flexibler was die Gestaltung der Anwendung betrifft und können deren verschiedene Funktionalitäten an den geeigneten Stellen einsetzen. Zur Illustration des modularen Ansatzes der Simulationsarchitekturen für virtuelle Umgebungen werden die zwei Frameworks vorgestellt.

2.2.1 FlowVR

Eine relevante Arbeit ist das Middleware Framework FlowVR [Allard u. a. 2004]. Dieses Framework ist aus Modulen und *Filtern* aufgebaut. Die Module enthalten die eigentlichen Softwarekomponenten, welche z.B. für die Berechnung einer Simulation oder für die Verarbeitung von Trackerdaten zuständig sind. Das ***A**pplication **P**rogram **I**nterface*[11] ist so gestaltet, dass die Umwandlung eines bestehenden Softwaremoduls in ein zu FlowVR kompa-

[11] kurz **API:** Ein definierter Satz von Routinen, Protokollen und Werkzeugen, um eine Softwareapplikation zu erstellen.

tibles Modul einfach umzusetzen ist. Ein solches Modul ist vereinfacht gesagt, eine periodische Schleife von Berechnungen, die Eingabedaten liest und neue Ausgabedaten produziert. Um direkte Abhängigkeiten zwischen den einzelnen Modulen einer Simulationsumgebung zu vermeiden, kann kein Modul ein anderes direkt ansprechen, d.h. die Module lesen nur die Daten, die an ihren *Eingabeports*[12] ankommen und schreiben ihre Ergebnisse auf ihre Ausgabeports. Das Modul-API basiert auf folgenden drei Funktionen:

1. Die `wait`-Funktion definiert den Übergang in einen neuen Iterationsdurchlauf. Sie stellt sicher, dass alle Module neue Eingabedaten an ihren Eingabeports enthalten.

2. Die `get`-Funktion ermöglicht es dem Modul die Nachricht, die an dem Eingabeport anliegt, anzunehmen.

3. Die `put`-Funktion erlaubt es dem Modul eine neue Nachricht auf dem Ausgabeport abzulegen.

Es kann jeweils nur eine neue Nachricht pro Port und Iteration geschrieben werden, und jede neue Nachricht von FlowVR wird automatisch mit einem Stempel versehen, der den Zähler des aktuellen Iterationsdurchlaufes enthält.

Sobald die an der Simulation beteiligten Module definiert sind, werden ihre Ein- und Ausgabeports durch sogenannte *Connections* verbunden, um einen **Datenfluss** herzustellen. Eine Connection ist ein *FiFo*-Kanal[13] eines bestimmten Typs, der genau einen Ursprung und ein Ziel hat, sowie festgelegt durch die Typisierung genau einen definierten Datentyp (beispielsweise *Integer*) transportieren kann. Jede Nachricht, die über einen solchen Kanal verschickt wird, besitzt einen Stempel mit der aktuellen Iterationsnummer und der Ursprungsidentifikation.

[12] **Ports:** Softwareinterne Schnittstellen, auf denen in diesem Fall Daten abgelegt werden können

[13] **FiFo:** kurz für First in First out. Bedeutet, dass die ersten Daten, die in eine Applikation hinein kommen, auch als erstes verarbeitet und wieder ausgegeben werden

Um die Möglichkeiten des FlowVR-Frameworks zu erweitern, wurden sogenannte **Filter** eingeführt. Diese besitzen ebenso getypte Ein- und Ausgabeports, jedoch können sie auf den Nachrichten, die sie zu verarbeiten haben, komplexe Operationen durchführen. Sie besitzen die Freiheit Nachrichten zu verwerfen, zu kombinieren, oder sogar gänzlich neue Nachrichten zu generieren. Ein weiteres wichtiges Merkmal, welches die Filter im Gegensatz zu den Modulen besitzen, ist die Möglichkeit mehr als nur eine Nachricht pro Port und Iterationsdurchlauf zu erhalten. Filter haben zudem freien Zugriff auf die Eingabepuffer und verarbeiten die Nachrichten im Normalfall basierend auf der Liste der Stempel mit Iterations- und Ursprungsidentifikation. Eine beispielhafte Anwendung für einen Filter könnte darin liegen, alle Nachrichten, welche ein Objekt betreffen, dessen *3D-Bounding-Box*[14] nicht innerhalb des aktuellen View-Volumes liegt, zu verwerfen, da es momentan nicht angezeigt wird.

Zur Synchronisation in FlowVR werden sogenannte *Synchronizer* verwendet, deren Aufgabe darin besteht, nicht-lokale Einschränkungen und Bedingungen aufzulösen. Synchronizer arbeiten auf Nachrichtenstempeln, was bedeutet, dass über alle ein- und ausgehenden Connections nur die oben erwähnten Stempel verschickt werden.

Normalerweise wird die Aktivität eines Synchronizers durch eingehende Nachrichten auf bestimmten Ports gesteuert. Da sie nur den Stempel einer Nachricht erhalten, und nicht den Teil, der die eigentlichen Daten enthält, sind sie generell mit *Filtern* verbunden. Diese Filter haben normalerweise zwei Eingabeports. Einen, der die komplette Nachricht eines Moduls erhält, und einen, der nur die Stempel von einem Synchronizer bekommt. Der Filter verarbeitet die eingehenden Nachrichten dann basierend auf den eingehenden Stempeln, die er von dem Synchronizer erhält.

[14] **3D-Bounding-Box:** Umhüllt ein Objekt, und gibt dessen relevante Größe für z.B. die Kollisionserkennung an.

Die Filter, inklusive dem Synchronizer, sind als dynamisch geladene Klassen (Plugins) implementiert. Jede FlowVR-Applikation wird durch ein spezielles Modul (*Controller* genannt) gesteuert. Dieses Modul wird automatisch beim Start der Applikation geladen. Er startet zunächst die Applikationsmodule über ihre eigenen Startkommandos, welche sich dann bei ihren *Daemons*[15] registrieren, die dann wiederum eine Bestätigung schicken. Sobald die Module geladen sind, schickt ihnen der Controller eine Liste mit den zu ladenden Plugins, um das FlowVR-Netzwerk aufzubauen. FlowVR benutzt eine XML-Beschreibung der Syntax der Startkommandos, welche mit den einzelnen Modulen assoziiert sind. Außerdem wird eine XML-Beschreibung des FlowVR-Netzes verwendet, welche eine explizite Beschreibung aller Komponenten enthält. Abbildung 2.9 zeigt eine beispielhafte Illustration eines FlowVR-Netzwerkes, mit Knoten, einem Filter und einem Synchronizer.

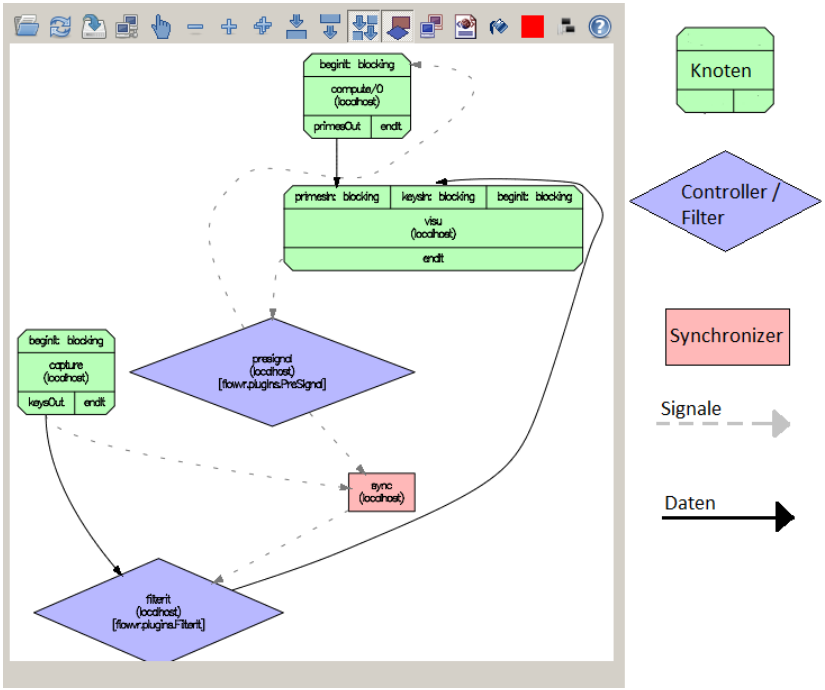


Abb. 2.9: Beispielhafte Illustration eines FlowVR-Netzwerkes. Übernommen von http://lear.inrialpes.fr

[15]**Daemon:** Ein Prozess, der zu einem definierten Zeitpunkt eine Aktion ausführt

Die Knoten (im Bild grün und achteckig dargestellt) werden mittels der erwähnten getypten Kanäle miteinander verbunden (im Bild als durchgehende Pfeile). Solche Verbindungen können auch zwischen Knoten und Filtern hergestellt werden, wie im Bild links zu sehen ist. Der Filter (im Bild blau und rautenförmig) erhält Daten von einem Knoten, die er gefiltert an einen anderen Knoten weitergibt. In der Abbildung zentral (ebenfalls als blaue Raute) ist der Controller dargestellt. Von diesem werden Signale (als gestrichelte Pfeile abgebildet) gesendet und empfangen. Unter anderem besteht eine solche Signalverbindung zum Synchronizer (rotes Viereck). Dieser synchronisiert Eingaben, Berechnungen und Filter.

2.2.2 Lightning

LIGHTNING ist ein modulares Simulationssystem, welches seit den späten 1990er Jahren in Stuttgart am Fraunhofer Institut für Arbeitswirtschaft und Organisation (auch kurz Fraunhofer IAO genannt) entwickelt wird [Blach u. a. 1998]. Es bietet einen offenen Entwicklungsrahmen für die Entwicklung interaktiver Anwendungen der Virtuellen Realität. Das System ist unter verschiedenen Betriebssystemen lauffähig. LIGHTNING stellt unterschiedliche Module bereit, welche dedizierte Aufgaben übernehmen. Zu diesen zählen zum Beispiel die Darstellung von 3D-Modellen oder die Verarbeitung von Ein- und Ausgabedaten. Das Framework stellt außerdem datenflussorientierte Kommunikationsmechanismen bereit. Der Anwendungsprogrammierer kann diese Module erzeugen und beliebig kombinieren.

LIGHTNING besitzt einen zweistufigen Aufbau, mittels dessen es ermöglicht wird, entweder zur Laufzeit über eine interpretierte Skriptsprache Einfluss zu nehmen oder zeitkritische Module direkt in C++ zu entwickeln. Neue Module können auch zur Laufzeit hinzugefügt werden. Dieses geschieht über einen standardisierten Kommunikationsmechanismus. Abbildung 2.10 zeigt die LIGHTNING Architektur. Neben den entwickelten Schichten ist (im Bild rechts) eine Kopplung generischer Module vorgesehen, die von der Lightning Bibliothek auf die Daten der Virtual Reality Applikation durchgreifen.

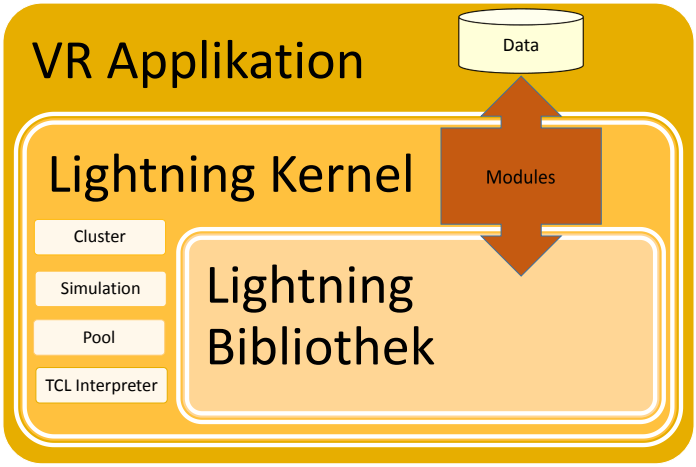


Abb. 2.10: Übersicht über die verschiedenen Schichten der LIGHTNING Architektur. Abbildung nachempfunden von http://www.imsys-vr.de

Der Lightning Kernel stellt verschiedene Funktionen bereit. Zu diesen zählen ein Cluster-Mechanismus, ein Simulationsmodul, ein Datenpool und TCL-Skriptinterpreter.

Bues u. a. [2008] haben eine Übersicht der Architektur, die Historie und zukünftige Ideen vorgestellt. Der Fokus liegt dabei auf dem Datenfluss-Modell des Frameworks. Das ursprüngliche Konzept des Datenflusses war inspiriert von dem in VRML eingesetzten Mechanismus. Zu diesem Zweck wurden Felder als Container für Werte sowie Objekte, die durch diese Felder beschrieben werden, entwickelt. Zwischen den Feldern verschiedener Objekte können Routen aufgebaut werden, die die Werte von einem Objekt zu dem nächsten propagieren. Während sich diese Art der Weitergabe von Werten für kontinuierliche Datenströme bewährt hat, produziert sie einen Überhang für sporadisch auftretende Events in Bezug auf die Datenmenge sowie den Aufwand der Programmierung. Neuere Versionen von LIGHTNING setzen auf eine Mixtur aus dem oben erwähnten Feldmechanismus für Datenströme, beispielsweise Daten eines Trackers, und prozeduralen Elementen für einzelne Ereignisse, wie zum Beispiel das Aktivieren einer Animation.

2.3 Actor Model

Im Gegensatz zu den bisher beschriebenen Systemen stellt das von Hewitt, Bishop und Steiger im Jahre 1973 vorgestellte Actor Model ein abstrakteres Konzept dar, das einen verteilten Ansatz propagiert [Hewitt u. a. 1973]. In einer modularen Architektur können die einzelnen Komponenten auch als "Aktoren" bezeichnet werden. Im Actor Model werden solche Aktoren als universelle Primitive für konkurrente digitale Berechnungen angesehen. Das heißt ein Aktor stellt für sich eine abgeschlossene Einheit dar, die in sich geschlossen, Berechnungen ausführen kann. Mehrere Aktoren arbeiten parallel, berechnen also konkurrent ihre digital vorliegenden Daten.

Das Actor Model ist inspiriert von vorhergehenden Konzepten wie dem *Lambda Kalkül* von Church [Church 1951] oder der Programmiersprache Smalltalk [Goldberg u. Robson 1983]. Das Lambda Kalkül kann als früheste *message passing* Programmiersprache angesehen werden. Die heutige Relevanz in der Informatik bezieht sich auf das sogenannte ungetypte Lambda Kalkül, welches unter anderem auch die ursprüngliche Inspiration für die funktionale Programmiersprache Lisp war. Weitere aktuelle Anwendungen des Modells werden in Abschnitt 2.3.4 vorgestellt, wobei unter anderem auf die Verwendung im Bereich der VR-Entwicklung (wie bei [Latoschik u. Tramberend 2012]) eingegangen wird.

Als fundamentales Konzept setzt das Actor Model die Philosophie **„Alles ist ein Aktor"** um. Dieses Prinzip ist ähnlich zu dem häufig in objektorientierten Programmiersprachen angetroffenen Konzept „Alles ist ein Objekt". Der Unterschied zwischen diesen beiden Philosophien liegt darin, dass objektorientierte Software in der Regel auch heutzutage noch sequentiell verarbeitet wird, während das Actor Model schon inhärent eine **parallele Verarbeitung** implementiert. Ein Aktor ist per Definition eine atomare Berechnungseinheit, welche in Reaktion auf empfangene Nachrichten nebenläufig folgende Aktionen ausführen kann:

1. **Senden** einer endlichen Anzahl von Nachrichten an andere Aktoren
2. **Kreieren** einer endlichen Anzahl neuer Aktoren
3. **Festlegen** des Verhaltens bei dem Empfang der nächsten Nachricht

Die oben gelisteten Aktionen haben hierbei keine designierte Reihenfolge und können parallel ausgeführt werden. Die Entkopplung des Senders von gesendeten Nachrichten war ein fundamentaler Fortschritt des Modells, durch den asynchrone Kommunikation und Kontrollstrukturen im message passing ermöglicht wurden. Empfänger von Nachrichten werden über sogenannte Adressen identifiziert – manchmal auch *mailing adresses* genannt. Ein Aktor kann nur Nachrichten an andere verschicken, dessen Adressen er besitzt. Diese kann er durch empfangene Nachrichten erhalten, oder sie kann ihm bekannt sein, falls der Empfänger ein Aktor ist, den er selbst kreiert hat. Das Modell ist also charakterisiert durch inhärente **Konkurrenz der Berechnungen**, welche innerhalb eines oder zwischen verschiedenen Aktoren stattfindet. Hierzu zählen dynamische Erzeugung neuer Aktoren, Inklusion von Adressen in Nachrichten und die ausschließliche Kommunikation durch direktes asynchrones message passing ohne Restriktion der Reihenfolge von empfangenen Nachrichten.

Abbildung 2.11 zeigt die Funktionsweise von Aktoren. Illustriert sind zwei Aktoren mit ihren Nachrichten-*Queues* sowie darin eingehende und bereits gepufferte Nachrichten. Diese werden dann zu der gegebenen Zeit verarbeitet, woraufhin interne Berechnungen vorgenommen werden. Am Ende des Prozesses steht die Ausführung der nächsten Aktion bzw. Antwort auf die erhaltene Nachricht. In der Abbildung werden weitere Nachrichten zwischen den Aktoren ausgetauscht. Außerdem ist am oberen Rand angedeutet, dass weitere Nachrichten von anderen nicht abgebildeten Aktoren eingehen können, welche auch verarbeitet werden müssen. Im Folgenden werden weitere grundlegende Eigenschaften des Modells betrachtet, die die Funktionsweise des Actor Models weiter beschreiben und dessen Aufbau verdeutlichen. Außerdem wird das Modell im Hinblick auf seine heutige Relevanz und Verwendung in anderen Systemen betrachtet.

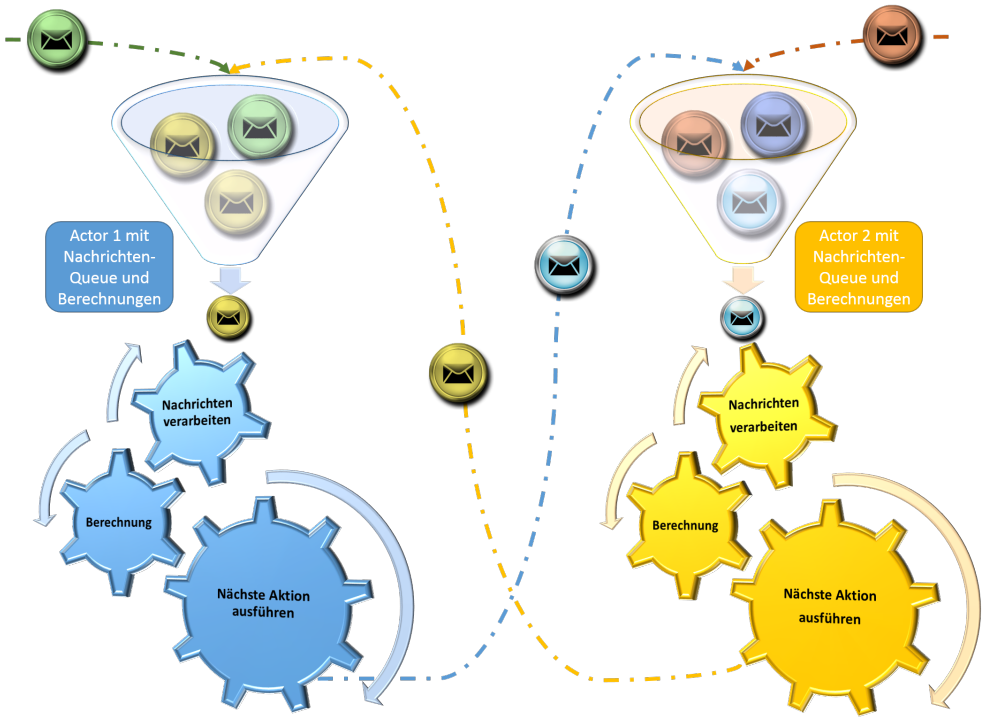


Abb. 2.11: Illustration der Funktionsweise des Actor Models. Es sind zwei Aktoren abgebildet, die in ihren internen Schleifen eingehende Nachrichten verarbeiten, Berechnungen anstellen und darauf basierend eine nächste Aktion ausführen. Im gezeigten Beispiel schicken sie dem jeweils anderen eine neue Nachricht.

2.3.1 Direkte Kommunikation und Asynchronität

Die innerhalb eines Aktor-Systems verschickten Nachrichten werden nicht zwangsweise gepuffert. Diese Tatsache stellt eine große Veränderung im Vergleich zu vorangegangenen Ansätzen konkurrenter Berechnungen dar. Die fehlende Pufferung der Nachrichten führte immer wieder zu Missverständnissen und ist auch heute noch ein streitbarer Punkt. Auch wenn die Nachrichten nicht aktiv zwischengespeichert werden, so kann doch argumentiert werden, dass sie in der Umgebung gelagert werden. Mitteilungen werden nichtsdestotrotz im Actor Model einfach verschickt (ähnlich IP-Paketen), ohne dass sie ein synchrones Händeschütteln mit dem Empfänger benötigen.

2.3.2 Variable Topologie

Eine Entwicklung des Actor Models war die Inklusion von Adressen in den versandten Nachrichten. Beeinflusst von *Packet Switching* Netzwerken schlug Hewitt mit dem Actor Model allerdings die Entwicklung eines neuartigen Systems vor, in welchem die Kommunikation keine zwingend benötigten Felder enthält. Das heißt in letzter Konsequenz, dass die Nachrichten auch komplett leer sein können. Wenn natürlich der Sender einer Nachricht darauf angewiesen ist, dass der Empfänger Zugriff auf Adressen bekommt, die ihm noch nicht vorliegen, müssen diese in der verschickten Nachricht enthalten sein. Ein Beispiel hierfür kann das Verlangen nach einer Antwort auf die gesendete Kommunikation sein. In diesem Fall enthält die Nachricht den gewünschten Inhalt begleitet von der Adresse eines anderen Aktors, der manchmal auch als *Continuation* oder *Stack Frame* bezeichnet wird. Über die Mitteilung dieser Adresse kann der Empfänger dann zum gegebenen Zeitpunkt eine entsprechende Antwort an diese verschicken.

Das Erzeugen neuer Aktoren kombiniert mit der Möglichkeit, Adressen in die Nachrichten mit einzubeziehen, hat zur Folge, dass eine **variable Topologie** zwischen den verschiedenen Aktoren bestehen kann.

2.3.3 Reihenfolge der Nachrichtenverarbeitung

Hewitt argumentierte gegen die Anforderung, dass Nachrichten in der Reihenfolge ankommen müssen, in der sie verschickt wurden. Sollte es der Fall sein, dass eine solche Notwendigkeit besteht, kann diese durch die Implementierung eines speziellen Queue Aktors realisiert werden, der seine Nachrichten in der Reihenfolge verarbeitet, in der sie bei ihm ankommen, also nach dem First In First Out Prinzip.

Es besteht im Actor Model also nicht die Notwendigkeit, dass eine Nachricht M1 von Aktor X an Aktor Y vor einer etwaigen Nachricht M2 von X an Y ankommt oder verarbeitet wird. So kann es beispielsweise vorkommen, dass während der Verarbeitung einer Nachricht M1 das Verhalten

induziert wird, zunächst eine Nachricht M2 zu bearbeiten, bevor mit M1 fortgefahren wird. Die Tatsache, dass diese Möglichkeit besteht, bedeutet allerdings nicht, dass ein solches Verhalten auch durchgeführt werden muss.

In diesem Punkt ähnelt das Aktor-Prinzip den Packet Switching Netzwerk-Architekturen, welche auf Grund dieser Tatsache die Möglichkeit beinhalten, Pakete zu puffern, multiple Pfade zum Verschicken der Pakete zu nutzen, defekte Pakete erneut zu senden und andere Optimierungen vorzunehmen.

Im Laufe der Zeit wurde auch die Möglichkeit betrachtet, Aktor-Systeme zu größeren zusammenzusetzen. Dieses stellt einen wichtigen Aspekt für große verteilte Systeme dar. Gul Agha hat diese Richtung in seiner Dissertation erforscht [Agha 1986], während die Idee später von weiteren Forschern aufgegriffen wurde, unter anderem von [Jung 1994].

2.3.4 Heutige Relevanz

Im Laufe der Jahre wurden einige formale Systeme entwickelt, welche Schlussfolgerung über Systeme im Actor Model erlauben. Hierzu zählen:

1. **Operationale Semantik**

   Die operationale Semantik ist eine Technik, um die Bedeutung von Programmen zu beschreiben. Die Bedeutung wird durch schrittweise Zustandsänderungen einer abstrakten Maschine definiert. Sie wird verwendet, um Eigenschaften nachzuweisen oder Programme zueinander in Beziehung zu setzen.

2. **Denotationelle Semantik**

   Die denotationelle Semantik beschreibt eine formale Semantik für eine formale Sprache. Die Sprache wird hierbei als mathematisches Modell verwendet, um eine echte Programmiersprache zu beschreiben. Konkret kann durch die Verwendung der denotationellen Semantik bei gegebener Belegung der Eingabevariablen das Endergebnis eines Computerprogramms berechnet werden.

Auch in vielen Applikationen kann das Actor Model als Modellierungsgrundlage verwendet werden. Hierzu gehören zum Beispiel:

- In **E-Mail-Systemen** können Accounts als Aktoren modelliert werden, während sie über ihre E-Mail-Adresse adressiert werden können.
- In **Web Services** können mit den Simple Object Access Protocol (kurz: SOAP) Adressen als Aktoren implementiert werden. Bei SOAP handelt es sich um ein Netzwerkprotokoll, mit dessen Hilfe Daten ausgetauscht sowie Funktionsaufrufe auf entfernten Systeme ausgeführt werden können.

Das Modell wurde in der Vergangenheit in einer Vielzahl von Programmiersprachen wie zum Beispiel Erlang, Io oder E implementiert. Es ist außerdem die Grundlage für moderne Programmiersprachen wie Stackless Python. Außerdem benutzen Computerspiele, die auf der Unreal Engine 3 basieren, die sogenannten Aktoren als Objekte in virtuellen Umgebungen. Hierzu liegen allerdings keine detaillierten Informationen vor, da es sich hierbei um kommerzielle und geschützte Anwendungen handelt. Die genannten früheren Programmiersprachen finden in heutigen VR-Anwendungen kaum Verwendung, was zu einem großen Teil daran liegt, dass keine explizite Verbindung zwischen diesen proprietären Sprachen und der Domäne der Virtuellen Realität hergestellt wurde. Um dieses zu erreichen, ist ein anderer Ansatz von Nutzen. Die Beschreibung der Prinzipien des Actor Models auf einer höheren semantischen Ebene ist nötig, um sich von konkreten Programmiersprachen zu lösen, welche nicht die Fähigkeiten mit sich bringen, effiziente und moderne Anwendungen zu entwickeln. Ein System, das die Fähigkeiten des Actor Models im Kontext der VR-Entwicklung ausnutzt, entsteht im Rahmen des SIRIS-Projekts an der Universität Würzburg in Zusammenarbeit mit der Beuth Hochschule für Technik Berlin. Dort wird basierend auf der Programmiersprache SCALA ein Demonstrator namens Simulator X entwickelt. SCALA basiert auf dem Aktor-Prinzip, sodass sich der Demonstrator die Möglichkeiten des Modells ebenfalls zu Nutzen macht. Nähere Informationen hierzu sind beispielsweise in [Fischbach u. a. 2011] und [Wiebusch u. a. 2012] zu finden.

2.4 Zusammenfassung

Dieses Kapitel hat einen Einblick in das Themengebiet der Virtuellen Realität gegeben, indem es sowohl die Historie wie auch die Gegenwart betrachtete. Es wurden Definitionen genannt, die das Feld aus unterschiedlichen Richtungen betrachten. Anschließend wurde in die Thematik der Simulationssysteme für VR-Anwendungen übergeleitet. Es wurden zwei verschiedene Prinzipien betrachtet – die **monolithischen** wie auch die **modularen** Systeme. Anhand von verwandten Frameworks wurden exemplarisch wichtige Aspekte beim Einsatz von VR-Systemen herausgestellt. Es ist festzuhalten, dass beide Ansätze Konzepte wie ein Rapid Prototyping, Möglichkeiten des Clustering oder der parallelen Berechnung unterstützen. Die modularen Systeme bieten im Vergleich zu den monolithischen Ansätzen allerdings eine höhere Flexibilität. Der modulare Aufbau eines Simulationssystems ist außerdem besser geeignet, um ein höheres Abstraktionsniveau der technischen Umsetzung von der Modellierung der Daten zu erreichen. Unter anderem deshalb soll auch im Zuge dieser Arbeit ein modularer Ansatz ausgebaut werden.

Nach der Betrachtung verschiedener konkreter Frameworks, wurde das Actor Model als ein abstrakteres Konzept vorgestellt. Beginnend mit einer Einführung in die Thematik, folgten die grundlegenden Funktionsweisen des Modells. Außerdem wurde auf die heutige Relevanz in der Informatik im Allgemeinen, wie auch im Kontext der Virtuellen Realität eingegangen. Die Integration des Aktor-Prinzips in ein modular aufgebautes Simulationssystem wird noch im weiteren Verlauf der Arbeit von Bedeutung sein (vgl. Kapitel 4).

Das nächste Kapitel beschäftigt sich mit der semantischen Modellierung virtueller Umgebungen. Dabei werden die Grundlagen semantischer Anreicherung virtueller Objekte behandelt sowie mit der semantischen Reflexion weitergehende Konzepte. Es wird auf ein semantisches Netz zur Wissensrepräsentation sowie auf das Allen-Kalkül eingegangen.

KAPITEL

3

GRUNDLAGEN SEMANTISCHER MODELLIERUNG

Um sich dem in Abschnitt 1.2 gesetzten Ziel einer abstrakten Modellierung der Simulationsinhalte sowie der Anwendungslogik zu nähern, werden im folgenden Kapitel wichtige Aspekte vorgestellt. Hierbei werden unter anderem bestimmte Vorgehensweisen, wie die Anreicherung virtueller Objekte um weitere Informationen sowie die **semantische Reflexion** virtueller Welten, vorgestellt. Darüber hinaus wird auf die Repräsentation von Wissen und der Möglichkeit dieses zu verarbeiten eingegangen. Außerdem wird ein generelles Konzept betrachtet, das die Definition zeitlicher Beziehungen erlaubt. Abgeschlossen wird das Kapitel mit einer Zusammenfassung, die die Relevanz der vorgestellten Punkte hervorhebt und in die vorgenommene Konzeptualisierung in Kapitel 4 überleitet.

Zunächst soll jedoch der für diese Arbeit zentrale Begriff der **semantischen Modellierung** definiert werden. Um sich der Bedeutung zu nähern, werden zunächst die beiden einzelnen Bestandteile des Begriffs betrachtet. Die Semantik ist ursprünglich ein Begriff, der dem Themengebiet der Sprachwissenschaften entstammt und wie folgt definiert wird:

Definition 3 (Semantik)**.** *Die Semantik ist ein [...] Teilgebiet der Linguistik, das sich mit den Bedeutungen sprachlicher Zeichen und Zeichenfolgen befasst [Dudenredaktion 2006].*

Im Themengebiet der Informatik wird der Begriff im Kontext der Bedeutung von Zeichenketten verwendet. Bei Programmiersprachen wird zwischen den Disziplinen der Syntax und der Semantik unterschieden. Die Syntax beschreibt die formale Struktur, der Programme folgen müssen, um gültig zu sein, während die Semantik die Bedeutung eines Programms oder Teile eines Programms beschreibt. Zu beachten ist, dass syntaktisch falsche Programme keine Semantik besitzen.

Der Begriff der Modellierung beschreibt das Erstellen eines Modells. Ein Modell kann allgemein nach dem deutschen Philosophen Albert Stachowiak [Stachowiak 1973] sinngemäß wie folgt definiert werden:

Definition 4 (Modell)**.** *Ein Modell definiert die Replikation eines Ausschnitts der Realität – also ein Abbild. Dabei kennzeichnen drei Merkmale das Verhältnis von Vorlage zu Abbild:*

1. ***Das Abbildungsmerkmal:*** *Das Modell ist stets nur ein Abbild der Realität und nicht identisch mit dieser.*

2. ***Das Verkürzungsmerkmal:*** *Modelle enthalten niemals die volle Beschreibung der Realität, sondern nur die für den Konstrukteur relevanten Elemente.*

3. ***Das pragmatische Merkmal:*** *Modelle sind ihren Originalen nicht eindeutig zugeordnet. Sie erfüllen lediglich eine Ersetzungsfunktion, die durch die Fragen nach dem Subjekt (**Für wen?**), nach dem Zeitpunkt (**Wann?**) und nach dem Zweck (**Wozu?**) strukturiert ist.*

Entsprechend der Definition dienen Modelle in der Informatik zur Abbildung eines Realitätsausschnitts, um eine Aufgabe mit Hilfe der Informationsverarbeitung zu lösen. In Kombination mit dem Begriff der Semantik wird die semantische Modellierung im Kontext der Entwicklung virtueller Umgebungen an dieser Stelle wie folgt definiert:

Definition 5 (Semantische Modellierung)**.** *Die semantische Modellierung bezeichnet das Erstellen eines Modells, das ein möglichst umfassendes Abbild der zu simulierenden virtuellen Umgebung darstellt. Dem Entwickler soll damit die Möglichkeit gegeben werden, virtuelle Umgebungen mit Blick auf die Bedeutung der einzelnen Bestandteile sowie möglichst unabhängig von speziellen Softwarekomponenten zu entwickeln. Die Modellierung der Simulation verfolgt dabei auf verschiedenen Ebenen die Fragen nach dem Subjekt, dem Zeitpunkt und dem Zweck.*

Semantische Modelle sind in der Informatik bei der Entwicklung relationaler Datenbanken weit verbreitet. Das dafür entwickelte *Entity-Relationship-Model* (kurz ERM) ermöglicht die Darstellung semantischer Informationen eines Weltausschnitts, mit dessen Hilfe im Anschluss eine entsprechende Datenbank entworfen werden kann. Im weiteren Verlauf werden die allgemeineren semantischen Netze verwendet, um ähnliche Strukturen zu erzeugen (vgl. Abschnitt 3.3).

Eine semantische Anreicherung grafischer Objekte ist ein Art Vorstufe der semantischen Modellierung. Auf diese wird im folgenden Abschnitt eingegangen.

3.1 Semantische Anreicherung virtueller Objekte

Die semantische Anreicherung von grafischen Objekten ist eine Möglichkeit, eine virtuelle Umgebung mit mehr Wissen über sich selbst zu modellieren. Hierbei werden die optisch dargestellten Objekte mit zusätzlichen Informationen versehen, welche sie zum Beispiel interaktiver machen. Das Hinzufügen weiterer Informationen zu grafischen Objekten kann auf verschiedenen Ebenen und in unterschiedlicher Granularität erfolgen. Bezugnehmend auf die verschiedenen Ebenen können solche semantischen Informationen beispielsweise in folgender Art hinterlegt werden:

1. Direkte Implementierung **im Quellcode** der programmierten Szene.

2. Modellierung von Knoten, die **im Szenengraph** unter die korrespondierenden grafischen Objekte eingefügt werden. Diese werden von Latoschik u. a. [2005] als semantische Entitäten[1] (auch *semantic entities*) bezeichnet. Mittels dieser Methode kann semantisches Wissen im Szenengraph verankert werden, sodass grafische Objekte durch weiteres Wissen erweitert werden können (vgl. Abbildung 3.1).

3. Vorhalten der semantischen Annotierung in einer **Wissensrepräsentation außerhalb** der Szenengraph-Struktur.

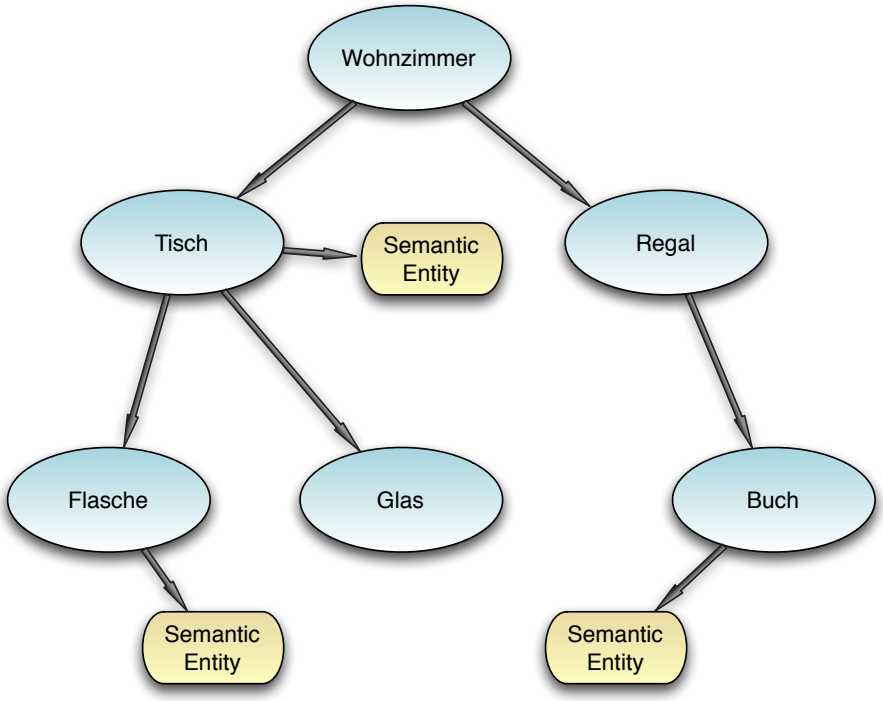


Abb. 3.1: Ein partiell mit semantischen Informationen angereicherter Szenengraph. Den grafischen Objekten werden mittels semantic entities, die im Graphen den entsprechenden Knoten zugeordnet werden, weitere Informationen hinzugefügt.

[1] **Entität** von lat. entitas „Ding“: Ein Grundbegriff aus der Philosophie, der etwas Seiendes bezeichnet.

Ebenso kann die Granularität der Informationen eine große Variabilität aufweisen. Hier reicht die Bandbreite von simplen semantischen Beschreibungen wie einer umgangssprachlichen Beschreibung des Objektes, um es beispielsweise über einen Spracherkenner referenzieren zu können, bis hin zu einer komplexen Beschreibung über etwaige Verformungseigenschaften oder möglichen Verbindungsstellen zu anderen Teilen in der virtuellen Umgebung.

Ein Beispiel für das Hinterlegen solcher Informationen ist in [Biermann u. a. 2007] beschrieben. Hier wird das Prinzip der semantischen Anreicherung virtueller Objekte auf ein virtuelles Konstruktionsszenario übertragen. Die Anwendung wissensbasierter Computergrafik findet in der virtuellen Konstruktion ein geeignetes Szenario. Im folgenden Abschnitt wird dieser Aspekt genauer betrachtet.

3.1.1 Virtuelle Konstruktion

Virtuelle Konstruktion nimmt in heutigen Produktionssystemen bereits einen hohen Stellenwert ein. Das Erstellen virtueller Entwürfe erspart den Entwicklern viel Zeit im Modellierungsprozess, wie auch enorme Kosten, die bei der Konstruktion realer Produkte entstehen. Das virtuelle Prototyping beschäftigt sich im Allgemeinen mit der Konstruktion von nicht-realen Prototypen und wird häufig in verschiedenen Bereichen des Maschinenbaus eingesetzt.

In der Automobilindustrie wird innerhalb des virtuellen Designprozesses die Handhabbarkeit neuer Einzelteile und Modelle sowie auch die Ergonomie neuer Fahrzeugcockpits erforscht und verbessert. Im VR-Bereich liegt der Fokus in Bezug auf das virtuelle Prototyping auf der Verbesserung der Interaktion mit den Konstruktionssystemen sowie auf dem intelligenten Zusammenspiel einzelner Bauteile und Komponenten. Eine beispielhafte Anwendung virtueller Konstruktionsszenarien findet sich mit der Virtuellen Werkstatt in Abschnitt 6.2.

Auf dem Gebiet der virtuellen Konstruktion werden in der Regel zwei Arten des Konstruierens unterschieden. Die erste beschäftigt sich mit der realitätsgetreuen externen Modellierung der realen Bauteile als virtuelle Bauteile, beispielsweise mit Hilfe von CAD[2]-Modellierungswerkzeugen, wie (vgl. [Loock u. Schömer 2001] oder [Zachmann u. Rettig 2001]). Diese Bauteile werden dann anschließend in der virtuellen Umgebung zusammengebaut, um sie im Zusammenspiel zu erproben. Der Nachteil an dieser Methode ist, dass die Teile meist statisch sind und nicht in der Anwendung direkt verändert werden können. Stellt man also ein Problem bei der Konstruktion des Gesamtsystems fest, muss das Bauteil erst wieder in den CAD-Werkzeugen überarbeitet werden, und der Designprozess beginnt von vorne. Eine sogenannte Variantenkonstruktion in Echtzeit ist mit dieser Art der Konstruktion nur sehr selten möglich.

Bei der zweiten Art der Konstruktion werden die virtuellen Bauteile mit semantischem Wissen angereichert, wie bei [Wachsmuth u. a. 1995], [Soto u. Allongue 2002] und [Peters u. Shrobe 2003] zu sehen ist. Die Verbindung von Methoden der Künstlichen Intelligenz und der Virtuellen Realität hat in der letzten Dekade immer mehr an Bedeutung gewonnen [Luck u. Aylett 2000] und wurde in verschiedene Richtungen erforscht. Cavazza [2000] schlägt für die Integration des semantischen Wissens eine gemeinsam genutzte Repräsentation vor, um die Wiederverwendbarkeit und den Grad der Adaptivität zu verbessern. Dieses Wissen umfasst sowohl intelligente Verbindungsstellen, als auch Informationen über mögliche parametrische Veränderungen der Teile. Die bei diesem Ansatz verwendeten Bauteile sind jedoch im Vergleich zu den oben erwähnten CAD-Modellen nicht so detailliert, beziehungsweise originalgetreu und erinnern eher an ein Baukastensystem. Allerdings bieten sie dem Benutzer deutlich mehr Interaktionsmöglichkeiten.

[2]**CAD:** kurz für Computer Aided Design. Im Deutschen mit Computergestütztem Entwurf zu bezeichnen

3.1.2 Anwendung im multimodalen Bereich

Neben der Anreicherung grafischer Objekte lässt sich das vorgestellte Konzept der semantischen Anreicherung in ähnlicher Art und Weise auch auf andere Modalitäten übertragen. Hierzu zählen sowohl gestische und sprachliche Eingaben, aber auch physikalisch simulierte Objekte oder akustische Ausgaben. Die Beschreibung einer Geste zur Interaktion kann ebenfalls semantisch hinterlegt werden, um so beispielsweise Objekte zu erzeugen oder ihren Zustand zu verändern. Um auf multimodale Art und Weise mit einem System interagieren zu können, ist ein Software-Mechanismus von Nöten, der die verschiedenen Instruktionen zu einer definierten Anweisung an das System zusammenfasst. Ein Beispiel für ein solches Verfahren ist in [Latoschik 2002] zu finden. Hier wurde für die Integration von Sprache und Gestik das sogenannte temporal Augmented Transition Network (kurz tATN) implementiert. Dieses erlaubt es, gleichzeitige Äußerungen von Sprache und Gestik miteinander in Einklang zu bringen. Der kombinierten Äußerung wird anschließend eine Bedeutung zugeordnet.

Um die Flexibilität multimodaler Eingaben zu erhöhen, ist neben der sprachlichen Instruktion eine Verarbeitung verschiedener Gesten notwendig. Durch die Nutzung verschiedener Gestentypen ist es dem Benutzer möglich, auf verschiedene Art und Weise mit dem System zu kommunizieren. Diese Typen schließen **deiktische**, **kinemimische** und **ikonische** Gesten ein. Im Folgenden sollen einige Details und Anwendungsbeispiele für diese verschiedene Typen gegeben werden, wobei der Fokus hier auf den ikonischen Gesten liegen wird.

Deiktische Gesten: Von *griech. Deixis* „zeigen“, bezeichnet der Begriff referenzierende Zeigegesten. Diese sind in der virtuellen Konstruktion vor allem für die Auswahl von Teilen von Bedeutung, welche nicht im direkten Greifraum platziert sind. Die Auswahl eines Teiles über deiktische Gesten ist immer mit einer sprachlichen Äußerung koordiniert, um das Teil gegebenenfalls noch weiter spezifizieren zu können (vgl. [Kranstedt u. a. 2006]).

Kinemimische Gesten: Angelehnt an *Kinematik* von *griech. Kinema* „Bewegung“, werden hiermit bewegungsbeschreibende Gesten bezeichnet. Diese werden verwendet, um ein virtuelles Objekt entsprechend der Geste zu bewegen. Durch die Übertragung der Bewegung kann beispielsweise die Rotation eines Objekts entsprechend der Handbewegung eines Benutzers realisiert werden. Hierbei werden sowohl die Rotationsachse der Bewegung, wie auch ihre Geschwindigkeit mit einbezogen. Die erkannte Trajektorie wird dabei entsprechend auf das virtuelle Teil übertragen. Wird zum Beispiel mit der Hand eine kreisförmig rotierende Bewegung ausgeführt, können auch Teile entsprechend der Geste rotiert werden, welche sich nicht im direkten Greifraum befinden [Latoschik u. a. 1999].

Ikonische Gesten: Von *griech. Ikon* „Bild“, werden mit dieser Bezeichnung formbeschreibende Gesten definiert. Durch die Verwendung von Imagistic Description Trees (kurz IDTs [Sowa u. Wachsmuth 2005]), welche eine Baumstruktur mit zusätzlichen Formbeschreibungen der Einzelteile enthalten, steht ein Repräsentationsformat mit hierarchischen Formbeschreibungen zur Verfügung. IDTs werden sowohl bei der Generierung neuer Teile, als auch bei der Äußerung einer ikonischen Geste des Benutzers aus den erkannten Formaspekten erstellt. Angewandt auf virtuelle Bauteile, beziehen sich die Formaspekte auf Längenabschnitte und Krümmungswinkel, welche entsprechend der Bauteilhierarchie mehrstufig ausgewertet werden können. Mit Hilfe einer Metrik, welche Ähnlichkeitswerte für Paare von IDTs berechnet, können unterschiedliche IDTs auf Ähnlichkeit, z.B. für die Referenzauflösung untersucht werden. Weitergehende Informationen zu der Verarbeitung ikonischer Formen und Gesten werden in Abschnitt 4.3.3 vorgestellt.

Eine Weiterentwicklung der semantischen Anreicherung virtueller Objekte ist das Prinzip der semantischen Reflexion, welches im folgenden Abschnitt erläutert wird.

3.2 Semantische Reflexion

Methoden der Künstlichen Intelligenz können eine wichtige Rolle innerhalb von Simulationssystemen spielen. Zum Einen stellen sie wichtige Funktionen bei der Pfadplanung von virtuellen Agenten oder der semantischen Beschreibung der künstlichen Welt dar. Zum Anderen spielen sie aber auch eine wichtige Rolle bei der Behandlung und Zusammenführung verschiedener Datenstrukturen unterschiedlicher Module, welche zu der Simulation beitragen. Unterschiedliche Komponenten arbeiten in der Regel auch mit verschiedenen Datenhaltungen. So arbeiten viele Grafiksysteme mit Hilfe eines Szenengraphen, während physikalische Simulationen keine solche Struktur aufweisen und nur einzelne Objekte (zunächst ohne Beziehung zu anderen) simulieren. Um ein einheitliches System zu erhalten, ist es notwendig, diese Strukturen zusammenzuführen. Zu diesem Zweck wurde von Latoschik u. Fröhlich [2007b] das Konzept der **semantischen Reflexion** entwickelt, das sich an folgende zwei Grundprinzipien anlehnt:

1. Das in Abschnitt 3.1 vorgestellte Prinzip der **semantischen Anreicherung** virtueller Objekte mittels semantic entities. Von diesem Prinzip wird die Möglichkeit übernommen, virtuelle Objekte mit zusätzlichem Wissen über sich und ihre Eigenschaften zu erweitern.

2. Das Prinzip der **Reflexion in Programmiersprachen**, das es ermöglicht, Instanzen von Typen dynamisch zu erzeugen, Typen an ein vorhandenes Objekt zu binden und Typinformationen von vorhandenen Objekten abzufragen. Außerdem können mit der Hilfe der Reflexion die Methoden vorhandener Objekte aufgerufen sowie auf ihre Felder und Eigenschaften zugegriffen werden.

Aus der Kombination dieser zwei Wirkungsweisen ergibt sich für die semantische Reflexion die folgende Definition:

Definition 6 (Semantische Reflexion)**.** *Die Semantische Reflexion definiert eine umfangreiche wissensbasierte Modellierung semantischer Informationen auf verschiedenen Ebenen. Dabei werden dynamische Zugriffe auf Informationen über Typ und Eigenschaften der verschiedenen Objekte zugelassen.*

Wie in Definition 6 festgelegt, umfasst die semantische Reflexion Objekte einer Simulation auf verschiedenen Ebenen. Dies schließt auch Entitäten ein, die nicht direkt sichtbar, hörbar oder fühlbar sind. Es werden alle Objekte auf einer gemeinsam genutzten Wissensrepräsentationsschicht abgebildet. Diese Schicht bietet auch den zentralen Zugriff auf alle Elemente. Dadurch können Objekte in der Szene verändert, neue Beziehungen zwischen Objekten hergestellt, Objekte hinzugefügt beziehungsweise gelöscht, oder die Topologie der Wissensrepräsentation verändert werden. Die **abgebildeten Entitäten** umfassen typischerweise folgende Daten:

1. Generelle Konzepte abgebildeter Szenenentitäten

2. Konkrete domänenabhängige Repräsentationen (Grafik, Audio, Physik, etc.)

3. Relevante Daten für die Interaktion mit Objekten (sprachliche oder gestische Abbildungen)

Neben diesen gängigen Daten werden auch Konzepte, die weniger sichtbar für den Endanwender sind, abgebildet. Es werden in der gemeinsamen Wissensrepräsentation zusätzlich Abläufe der Anwendungslogik und des Anwendungsdesigns reflektiert. Die genauen Abläufe und Konzepte werden im weiteren Verlauf der vorliegenden Arbeit näher erläutert. Die Integration des Prinzips der semantischen Reflexion in ein konkretes Simulationssystem wird in Abschnitt 4.2.3 anhand einer konzeptuellen Umsetzung dargestellt.

Zusammenfassend lässt sich an dieser Stelle sagen, dass die semantische Reflexion einige Vorteile mit sich bringt. Dies gilt vor allem im Hinblick auf eine **einheitliche Repräsentation des vorhandenen Wissens**, sowohl für die simulierte Szene, wie auch für die weniger sichtbaren Informationen der Anwendungslogik.

3.3 Semantische Netze als Wissensrepräsentation

Semantische Netze gehen auf den Sprachwissenschaftler Ross Quillian zurück. Dieser hat sie in den frühen 1960ern als Repräsentationsform für semantisches Wissen vorgeschlagen [Quillan 1966]. Sie eignen sich hervorragend zur Repräsentation von **hierarchisch strukturiertem Wissen**. Weitere Informationen zu semantischen Netzen finden sich zum Beispiel in [Charniak 1985].

Abbildung 3.2 zeigt beispielhaft, wie Wissen in semantischen Netzen strukturiert werden kann. Ausgehend von sehr grundlegenden Konzepten werden spezielle Formen abgeleitet, von denen in einer weiteren Hierarchie-Stufe Instanzen angelegt werden können. Die `is-a` (Ist-ein) und `inst-of` (Instanz-von) Relationen stellen diese Bezüge her und bieten somit die Grundlage für die Struktur in semantischen Netzwerken. Es können allerdings nicht nur solche hierarchischen Bezüge definiert werden, sondern vielmehr können den einzelnen Knoten auch durch den Einsatz weiterer Relationen Attribute hinzugefügt werden. Diese Attribute, wie zum Beispiel die Eigenschaft, dass Menschen einen Kopf haben, werden an die speziellen Instanzen vererbt und besitzen damit für alle Instanzen eine generelle Gültigkeit.

Die in dieser Arbeit verwendete Wissensrepräsentationsschicht basiert auf einem funktional erweiterbaren semantischen Netzwerk [Latoschik u. Schilling 2003]. Mittels dieses Netzwerkes werden alle notwendigen Daten abgebildet. Es ist in verschiedene sogenannte Domänen unterteilt, welche verschiedene Aspekte und Schichten der Simulation abbilden. Die spezialisierten Abbildungen innerhalb der Domänen werden mit den generellen Entitäten verknüpft. Die **funktionale Erweiterbarkeit** des Netzes erlaubt es, über die Konzepte eines herkömmlichen semantischen Netzes hinaus, Knoten um Funktionen zu erweitern, welche bestimmte Relationen inhaltlich abbilden. Stellt man sich beispielsweise vor, dass in einer grafisch abgebildeten Szene ein Objekt auf dem anderen liegt, soll im Falle einer Bewegung des zu

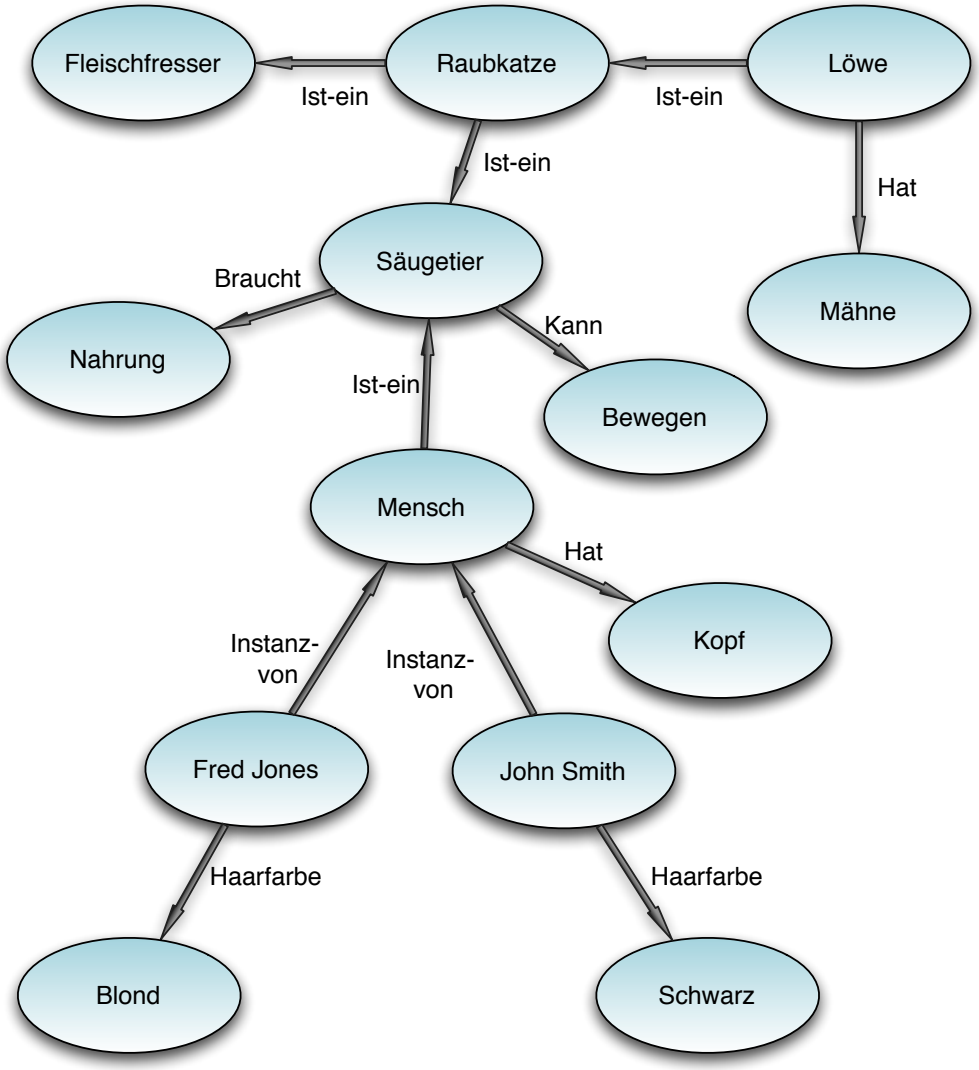


Abb. 3.2: Beispielhafte Abbildung eines semantischen Netzes.

Grunde liegenden Objektes auch das aufliegende mit bewegt werden. Solche Beziehungen werden in Szenengraphen normalerweise durch eine hierarchische Gruppierung der Objekte realisiert. Um solche Eigenschaften auch auf dem vorliegenden Netz reflektieren zu können, wird im Falle einer Instanziierung der dafür bestimmten Relation im semantischen Netz auch direkt die benötigte Szenengraph-Struktur in der Grafikkomponente angelegt.

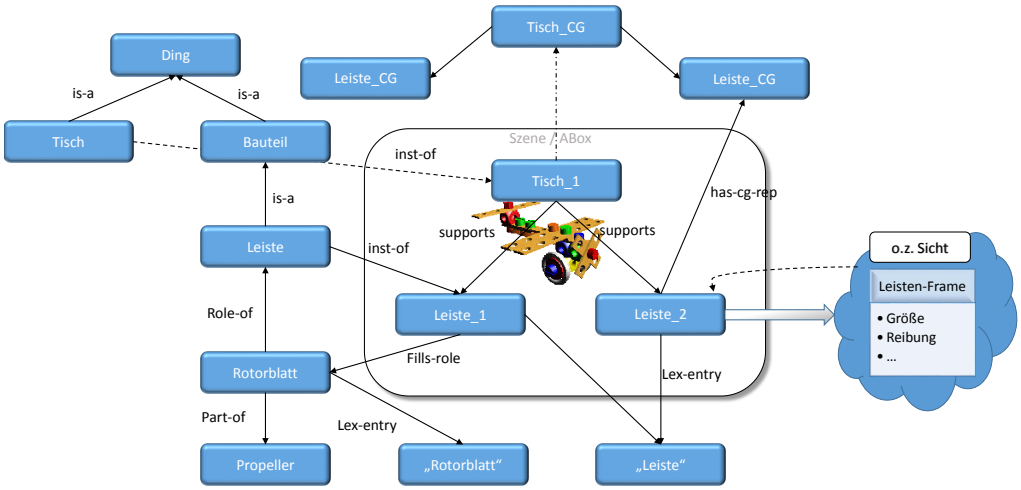


Abb. 3.3: Beispielhafte Abbildung eines funktional erweiterbaren semantischen Netzes. Nachgebildet aus [Heumer u. a. 2005]

Das **funktional erweiterbare semantische Netz** unterscheidet sich außerdem von anderen semantischen Netzen, durch eine objektzentrierte Sicht. In dieser werden Knoten als Objekte gesehen. Während in herkömmlichen semantischen Netzen Attribute an alle abgeleiteten Knoten vererbt werden, werden in den hier verwendeten Knoten Attribute in Slots gespeichert, welche den Slots in einer Framestruktur gleichen. Die Vererbung solcher Attribute kann explizit definiert werden und muss nicht zwangsweise an alle ableiteten Konzepte und Instanzen weitergegeben werden.

Abbildung 3.3 zeigt eine beispielhafte Anwendung des funktional erweiterbaren semantischen Netzwerkes anhand einer kleinen Szene, die einen Tisch und zwei darauf liegende Leisten enthält, die beim Zusammenbau einen Propeller darstellen. Während links im Bild eine generelle Abbildung der hierarchischen Wissensstruktur abgebildet ist, ist in der Mitte die konkretisierte Szene dargestellt. Darüber liegend wird die grafische Struktur gezeigt. Am rechten Rand der Grafik ist die Framestruktur angedeutet, die die Attribute der abgebildeten Leiste enthält.

3.4 Semantische Traverser

Die Traversierung von Baumstrukturen ist in der Informatik schon lange ein probates Mittel, um effizient mit solchen Datenstrukturen arbeiten zu können. In der Praxis wird hierbei jeder Knoten des Baums genau einmal durchlaufen und eine definierte Aktion ausgeführt. Gebräuchliche Aktionen sind hierbei:

- Die **Suche** nach einem bestimmten Knoten
- **Sammeln** von Informationen
- **Ändern** von Knoten-Attributen
- **Hinzufügen oder Löschen** eines einzelnen Elements oder eines ganzen Teilbaums

Das Traversieren von Bäumen wird durch die Reihenfolge klassifiziert, in der die Knoten des Baums besucht werden. Die zwei grundlegenden Operationen sind die Breiten- und Tiefensuche.

Im Gegensatz zum einfachen Fall der Baumstrukturen, wird bei der Traversierung von Graphen jeder Knoten gegebenenfalls mehrfach besucht. Außerdem ist bei Graphen nicht gewährleistet, dass ein Knoten existiert, der als Wurzel fungiert und mit allen anderen Knoten über einen direkten Weg verbunden ist. Die Traversierung von Graphen ist also aufwändiger als die von Bäumen. Die zwei grundlegend unterschiedlichen Techniken sind ebenfalls die Tiefen- und Breitensuche. Bei einer Breitensuche in Graphen werden zunächst die „Geschwister-Knoten“ besucht, bevor die untergeordneten „Kind-Knoten“ durchlaufen werden. Typischerweise wird bei dieser Traversierung eine *Queue* als Datenstruktur verwendet. Die Tiefensuche agiert genau umgekehrt. Es werden erst die Kinder und dann die Geschwister besucht. Als Datenstruktur wird hierbei oft ein *stack* verwendet.

Auch in der Domäne der Computergrafik werden solche Operationen in Verbindung mit Szenengraphen angewandt. Die Traversierung von Szenengraphen ist unabdingbar für die grafische Darstellung der Szene. Folgende Punkte werden oft verwendet:

- **Sammeln** von benötigten Informationen
- **Abbilden** von lokalen auf globale Koordinaten
- **Modifizieren** von Attributen, zum Beispiel für Animationen

Die Traversierung von Szenengraphen erfolgt oft von oben nach unten und von links nach rechts. Genau diesen Fall zeigt auch die Grafik 3.4

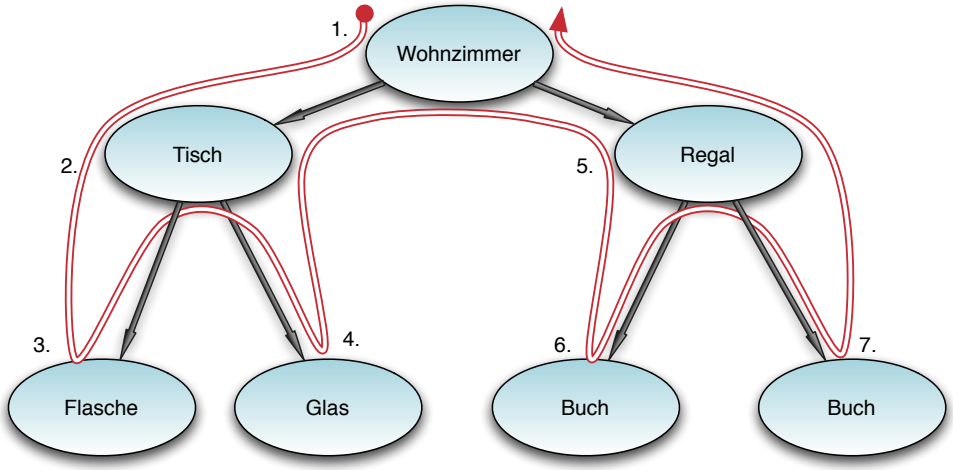


Abb. 3.4: Beispielhafte Traversierung eines Szenengraphen

Semantische Traverser [Peuser u. Latoschik 2008] operieren auf dem semantischen Netz und suchen designierte Knoten und Relationen. Haben sie ihr gesuchtes Ziel gefunden, werden als Reaktion bestimmte definierte Aktionen ausgeführt. Das Anwendungsgebiet solcher Traverser kann sehr vielfältig sein. Die Bandbreite reicht von der Ausführung sehr einfacher Operationen wie dem Löschen eines einzelnen Knotens bis hin zur Konstruktion und Ausführung komplexer Funktionsaufrufe wie sie in Abschnitt 4.3.3 erläutert wer-

den. Semantische Traverser sind die zentralen Routinen, die auf dem semantischen Netz operieren. Neben dem Suchen nach Knoten und dem Ändern ihrer Attribute sind sie außerdem in der Lage, neue Relationen und Knoten innerhalb des semantischen Netzes zu erzeugen. Das heißt, die Traverser können neues Wissen erzeugen und vorhandenes abstraktes Wissen durch das Erstellen von Relationen miteinander verknüpfen. Um beispielsweise Knoten in zeitliche Relation zueinander zu setzen, können Verknüpfungen erstellt werden, die eine temporale Semantik abbilden und dabei Allens Intervallalgebra folgen, welche im nächsten Abschnitt erläutert wird.

3.5 Allens Intervallalgebra

Bei Allens Intervallalgebra, auch als Allen-Kalkül bezeichnet, handelt es sich um eine Form der Logik, die zur Repräsentation von **zeitlichen Zusammenhängen** sowie zum logischen Schließen verwendet wird. James F. Allen stellte seine Algebra im Jahr 1983 vor [Allen 1983]. Allen definiert in seiner Algebra mögliche zeitliche Zusammenhänge zwischen Intervallen und beschreibt einen Algorithmus, um basierend auf diesen zeitlichen Relationen zwischen Ereignissen Schlüsse ziehen zu können. Diese Zusammenhänge sind in 13 temporale Relationen strukturiert (siehe Abbildung 3.5).

Relation	Illustration	Interpretation
X < Y Y > X	X Y	X findet vor Y statt
X tY Y tX	X Y	X trifft Y
X uY Y uX	X Y	X überlappt Y
X sY Y sX	X Y	X startet mit Y
X wY Y wX	X Y	X findet während Y statt
X bY Y bX	X Y	X endet mit Y
X = Y	X Y	X ist gleich Y

Abb. 3.5: Allens 13 zeitliche Relationen im Überblick

Die ersten sechs Relationen können ebenso invers beziehungsweise bidirektional betrachtet werden. Dadurch werden insgesamt 13 Relationen definiert. Durch die Verwendung der Intervallalgebra können Aktionen in eine zeitliche Abfolge in Relation gesetzt werden. Die vorgeschlagenen Relationen bieten die Möglichkeit eine Vielzahl denkbarer Abfolgen abzubilden. Betrachtet man den Satz: *„John liest die Zeitung während des Frühstücks. Im Anschluss daran geht er zur Arbeit.“*, können beispielsweise folgende Anwendungen der Intervallalgebra dargestellt werden.

(1) Lesen (w, s, b) Frühstück
(2) Frühstück (<, t) Arbeit

Die drei Relationen (w, s, b) aus (1) beschreiben dabei den exakten temporalen Zusammenhang. Es existiert ein gemeinsamer Start- und Endpunkt der zeitgleich stattfindenden Aktionen *Lesen* und *Frühstücken*. Die Relation (t) aus (2) beschreibt die Tatsache, dass direkt im Anschluss an das Frühstück, die Arbeit folgt. Zusätzlich verdeutlicht die (<)-Relation, dass das Frühstück zuerst stattgefunden hat.

Das Allen-Kalkül definiert außerdem auch eine Kompositionstabelle. Diese ermöglicht anhand von gegebenen Relationen zwischen X und Y und zwischen Y und Z auf die Relation von X und Z zu schließen. Angewandt auf das oben stehende Beispiel kann also geschlussfolgert werden, dass die Aktion *Lesen* vor der *Arbeit* stattfand:

(3) Lesen (t, <) Arbeit

Das beschriebene Beispiel ist in Abbildung 3.6 anschaulich in einer Übersicht dargestellt. Diese formale Beschreibung zeitlicher Abfolgen eignet sich gut für den Einsatz in virtuellen Umgebungen, wenn es darum geht, **temporale Zusammenhänge** zwischen Ereignissen abzubilden. Eine nähere Betrachtung der konkreten Umsetzung findet sich in Abschnitt 4.3.3.

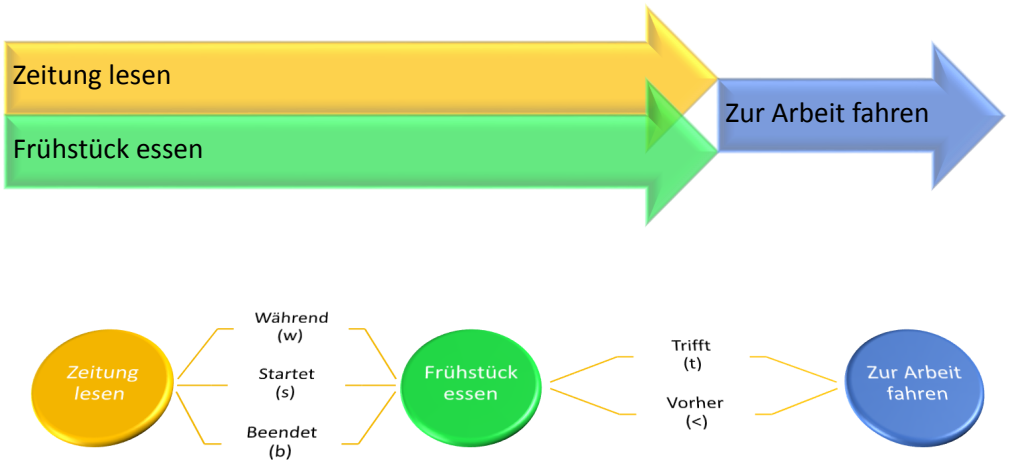


Abb. 3.6: Das beschriebene Beispiel in einer Übersicht. Die Aktionen werden wie beschrieben mit den entsprechenden Relationen aus Allens Intervallalgebra verknüpft.

3.6 Zusammenfassung

Nach der Einführung in die Thematik mit der Definition des Begriffs **semantische Modellierung** wurden die Grundlagen semantischer Anreicherung virtueller Objekte vorgestellt. Dabei wurde das Prinzip erläutert sowie die Anwendung in den zwei Bereichen der virtuellen Konstruktion und der Verarbeitung multimodaler Informationen vorgestellt. Die vorgestellten Anwendungsszenarien sind im weiteren Verlauf noch von Bedeutung. In Kapitel 6 wird ein Konstruktionsszenario vorgestellt, das dem entwickelten Framework als Demonstrator dient. Nach der semantischen Anreicherung wurde die semantische Reflexion als weitergehendes Prinzip eingeführt, das es erlaubt, alle an einer Simulation beteiligten Informationen semantisch zu reflektieren. Als Repräsentationsstruktur dient dazu das vorgestellte funktional erweiterbare semantische Netz, welches im Anschluss vorgestellt wurde. Abschließend wurde Allens Intervallalgebra dargelegt. Diese erlaubt es temporale Beziehungen zwischen Objekten und Aktionen zu definieren.

Die Integration der dargelegten Prinzipien wird im folgenden Kapitel 4 vorgestellt. Dieses wird in zwei Bereiche aufgeteilt, die sich mit der modularen Architektur und der semantischen Modellierung beschäftigen. Dabei werden Anforderungen, eine konzeptuelle Umsetzung und eine exemplarische Integration verschiedener Strukturen dargestellt.

KAPITEL

4

KONZEPT UND INTEGRATION

Das nachfolgende Kapitel zeigt, wie die in den Kapiteln 2 und 3 vorgestellten Modelle und Konzepte in die vorliegende Domäne der semantischen Modellierung virtueller Umgebungen integriert werden. Es wird gezeigt, wie diese ihren Beitrag zu der Umsetzung eines modularen Virtual Reality Frameworks leisten, wie sich dieses aufbaut und aus welchen Komponenten es sich zusammensetzt. Dabei wird auf die zwei Bestandteile dieser Arbeit eingegangen – den Aufbau eines **modularen Simulationsframeworks** und die Abstraktion der Simulation mittels der **semantischen Modellierung**. Für beide Abschnitte werden Anforderungen definiert, ein Konzept ausgearbeitet und die Integration verschiedener Aspekte vorgestellt. Die durchgeführten Arbeiten verfolgen die in Abschnitt 1.1 festgelegte Motivation, eine **längere Persistenz** von VR-Anwendungen zu erreichen. Teile der folgenden Ausarbeitung wurden bereits in Latoschik u. Fröhlich [2007b], Latoschik u. Fröhlich [2007a], Fröhlich u. Latoschik [2008] und Fröhlich, Biermann, Latoschik, u. Wachsmuth [2009a] veröffentlicht.

Die Ausarbeitung der verschiedenen Konzepte basiert auf dem Simulationsframework SCIVE (Simulation Core for Intelligent Virtual Environments). Teile dieser Architektur wurden von Latoschik, Fröhlich, u. Wendler [2006] veröffentlicht und im Zuge dieser Dissertation weiterentwickelt. Bevor

die oben stehende Ausarbeitung der verschiedenen Teilbereiche vorgenommen wird, wird auf die technische Basis eingegangen.

4.1 Technische Ausgangsbasis

Als technische Grundlage dient das modulare Simulationsframework SCIVE. Abbildung 4.1 zeigt den generellen Aufbau der vorliegenden Architektur in einer einfachen schematischen Form.

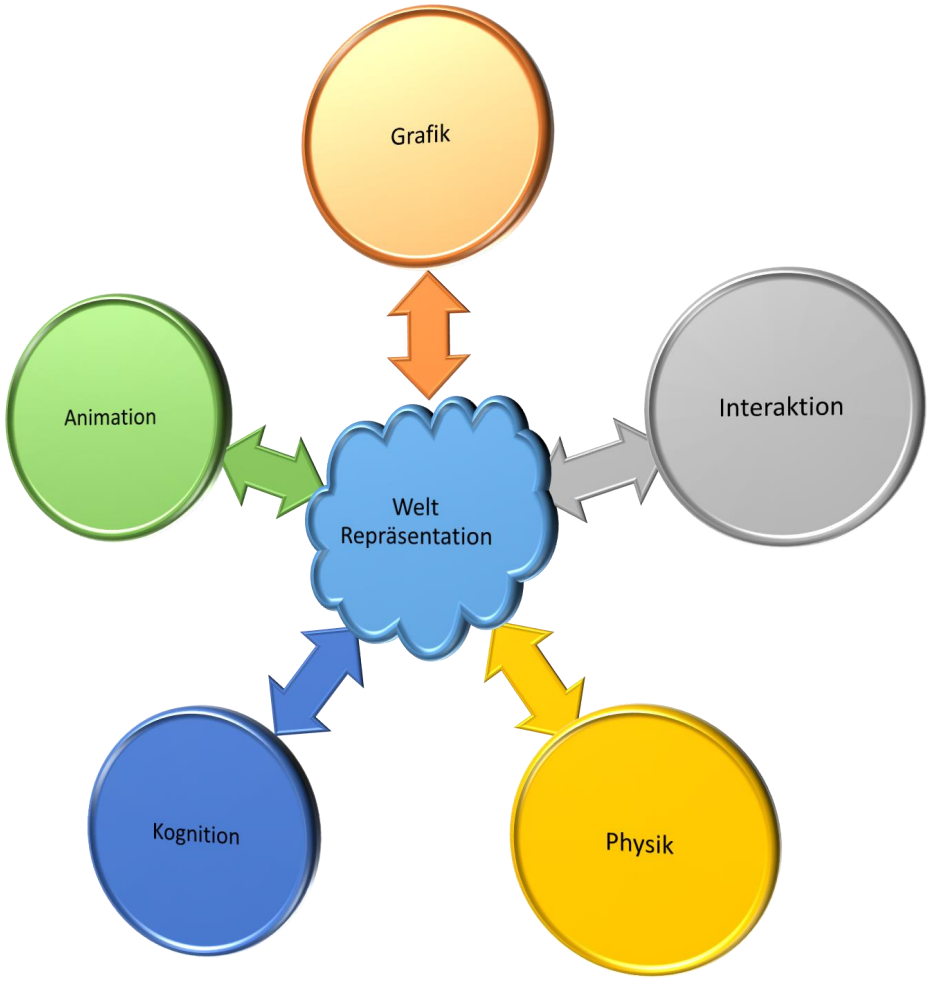


Abb. 4.1: Schematische Darstellung der SCIVE Architektur mit fünf Modulen (vgl. auch [Latoschik u. Fröhlich 2007b])

Abgebildet sind fünf unterschiedliche Module, welche zur Berechnung der Simulation beitragen und ihre Ergebnisse in einem globalen Weltzustand integrieren. Die abgebildete Grafik zeigt eine Architektur, welche als zentrales Element die aktuelle **Welt-Repräsentation** verwendet. Alle andere Module stellen dieser ihre Ergebnisse zur Verfügung. Beim Betrachten eines Beispiels fallen aber einige Nachteile auf. Angenommen, die abgebildete Architektur wird für die Simulation eines Agenten in einer virtuellen Welt verwendet. Hierfür wird zunächst ein Modul zur grafischen Ausgabe des Agenten und seiner Umgebung benötigt. Darüber hinaus sind für eine realistische Simulation noch weitere Module von essentieller Bedeutung. Es werden Komponenten für die Simulation der kognitiven Fähigkeiten des Agenten, seiner Animation, der physikalischen Simulation der Umgebung sowie für die Benutzerinteraktion benötigt. Der Agent sollte in seine virtuelle Umgebung eingebettet sein, was bedeutet, dass er in der Lage sein muss, diese wahrzunehmen, und dass seine Handlung glaubhafte Effekte hervorrufen.

Bei der Betrachtung der in Abbildung 4.1 abgebildeten Architektur, in Hinsicht auf die glaubhafte Simulation des beschriebenen Beispiels, fallen folgende Punkte auf. Die **Kognitions-Komponente** braucht eine enge Verbindung zu dem herrschenden Weltzustand, um es dem Agenten zu ermöglichen, seine Umgebung möglichst schnell wahrnehmen und realistisch darauf reagieren zu können. Es muss also eine virtuelle Manifestation perzeptiver Organe (zum Beispiel virtuelle Augen) des Agenten vorliegen.

Die übrigen Komponenten können spezifische Entitäten (inklusive des Agenten) in der Welt verändern. Entitäten sollten in dem vorliegenden Beispiel solange unter Kontrolle der physikalischen Simulation bleiben, bis sie entweder einer Änderung durch das externe Animationsmodul, oder einer Interaktion durch den Benutzer unterliegen. Abhängig vom Zustand der Applikation muss der Modulzugriff nun entsprechend dem gewünschten Verhalten geregelt werden. Der Agent soll solange durch seine kinematische Animation bewegt werden, bis er auf ein physikalisches Hindernis trifft. Wenn nun eine solche Kollision auftritt, muss der Zugriff auf die Animation des Agenten

zwischen der kinematischen Animation und der physikalischen Simulation geregelt werden, um glaubhafte Animationen während des Zusammentreffens zu generieren, bis die Kinematik wieder die Kontrolle über die Bewegungen übernimmt.

Die verwendete VR-Simulationsarchitektur sollte es erlauben zu definieren, wann und wie bestimmte Module die Kontrolle über einzelne Entitäten bekommen. Außerdem sollte es in einer modularen Architektur möglich sein, Komponenten mit minimalen Aufwand auszutauschen. Hieraus resultieren nun folgende **Anforderungen an die Simulationsarchitektur:**

1. Jedes Modul muss prinzipiell in der Lage sein, Daten mit jedem anderen Modul austauschen zu können, seien dies nun nur einzelne Attribute einer Entität oder komplette Weltzustände.

2. Der Datenfluss muss zwischen den Modulen dahingehend kontrolliert werden, dass es immer einen kohärenten Weltzustand gibt.

In einer solchen Architektur verliert die **Datenhaltung** ihre zentrale Position und wird behandelt wie jedes andere Modul. Somit hat die Datenhaltung eine eigene Repräsentation, welche mit den übrigen Modulen synchron gehalten werden muss. Eine solche Architektur ist in Abbildung 4.2 dargestellt.

SCIVEs Architektur stellt die grundlegenden Mechanismen bereit, um intelligente echtzeitfähige Applikationen zu erstellen. Sie erlaubt es, Module miteinander zu verbinden, wobei diese lose oder eng gekoppelt sein können. Im ersten Fall trägt das fragliche Modul nur von Zeit zu Zeit zur gesamten Simulation bei und beeinflusst unter Umständen nur einzelne Werte, im zweiten Fall hat das Modul Zugriff auf alle Entitäten und alle Attribute und ändert sie in jedem Simulationsschritt der Applikation. Die Module sind konzeptuell durch eine Instanz verbunden, welche ihre zeitliche Synchronisation regelt, und zwei Hauptaufgaben erfüllt:

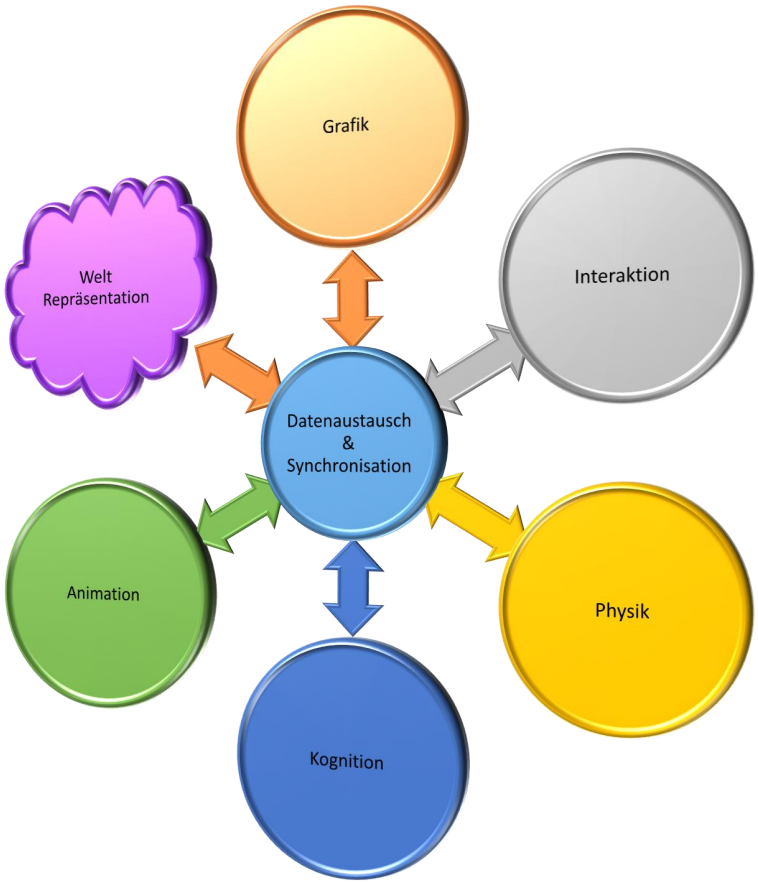


Abb. 4.2: Schematische Darstellung der Architektur mit einer zentralen Organisationskomponente (vgl. auch [Latoschik u. Fröhlich 2007b]).

1. Sie ist für das sogenannte *Bootstrapping* des Systems verantwortlich, bei dem die verschiedenen Weltrepräsentationen der Komponenten geladen werden, um einen **initialen kohärenten Weltzustand** zu gewährleisten.

2. Sie stößt zur Laufzeit die diversen Simulationsschleifen der Module asynchron an, **kontrolliert** die anschließende Sammlung der Daten, mögliche Konfliktauflösung und die Weiterverteilung der Daten.

Dieses Vorgehen bringt einen signifikanten Vorteil im Bezug auf die parallele Verarbeitung (mit Benutzung von mehreren Kernen, CPUs oder Rech-

nern). Der leicht höhere Aufwand in Bezug auf Rechenzeit und -kapazität, der durch die Synchronisationsmechanismen entsteht, wird in diesem Falle zu Gunsten des modularen Aufbaus akzeptiert. Die **Datenaustausch-Komponente** übersetzt Werte verschiedener Repräsentationen, welche in den Modulen als gleiche Werte vorliegen. Abbildung 4.3 zeigt eine detailliertere Darstellung des Datenaustausches zwischen den Modulen. Die einzelnen Daten werden in den Komponenten eingesammelt und über die zentrale Austauschkomponente an die anderen Module propagiert.

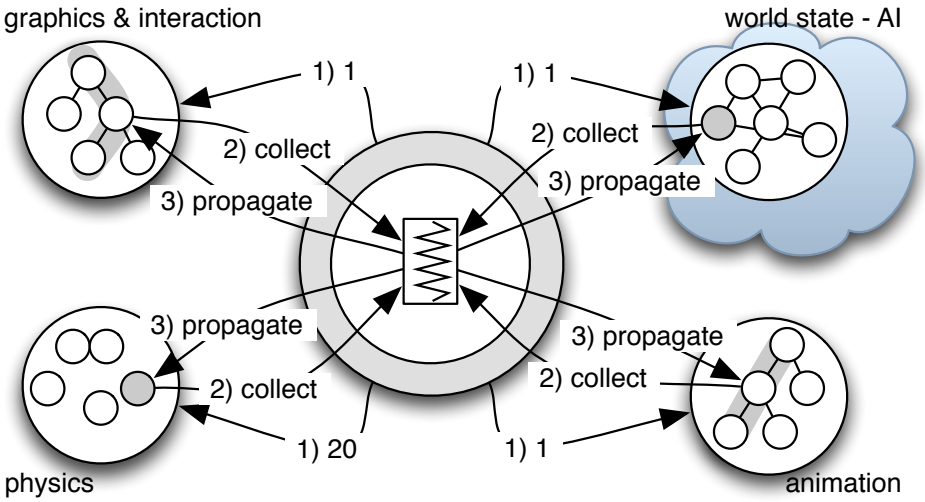


Abb. 4.3: Schematische Darstellung der Simulationsarchitektur mit dazugehörigem Datenaustausch [Latoschik u. Fröhlich 2007b]

Sogenannte **Filter** regeln den Zugriff von Modulen auf Entitäten, um konfligierende Ergebnisse aufzulösen. Dieses ist aus dem Grunde notwendig, da in einem System ohne solche Mechanismen immer das letzte Modul, das seine Daten den anderen zur Verfügung stellt, die Oberhand behält, indem die übrigen Ergebnisse überschrieben werden. Es muss also geregelt werden, wie bestimmte Module die einzelnen Entitäten beeinflussen. Auf die dafür notwendigen Mechanismen wird im folgenden Abschnitt eingegangen.

4.1.1 Kontrollfluss

Der **nebenläufige Zugriff** auf Entitäten in der zentralen Wissensbasis kann zu inkonsistenten Ergebnissen des Weltzustandes der einzelnen Module führen, wenn der Zugriff nicht in irgendeiner Art und Weise geregelt wird.

SCIVE stellt einen speziellen Datenpropagierungsgraphen bereit, welcher ein System von Verbindungen und Filtern verwendet. Filter empfangen Events, für die sie sich registriert haben. Sie können Berechnungsergebnisse verarbeiten, welche bestimmte Attributwerte betreffen und daraufhin neue Signale produzieren. Zusätzlich können Filter anstatt Events zurück zu geben, bestimmte Aktionen in den Simulationsmodulen anstoßen. So können zum Beispiel zusätzliche Kräfte in der Physiksimulation hinzugefügt werden. Bestimmte Änderungen können auf verschiedene Art und Weise implementiert werden. Die Verschiebung eines Objektes kann beispielsweise durch das Setzen der Position aus der Interaktionskomponente oder alternativ durch die Addition physikalischer Kräfte realisiert werden. In einer einfachen Konfiguration können Filter den Wert einer Entität allein durch die physikalische Simulation berechnen lassen, während ein anderes durch eine vorberechnete Skelettanimation oder durch eine Form von Benutzerinteraktion bestimmt wird. In einem komplexeren Szenario können auch komplett neue Werte berechnet werden, indem sie von verschiedenen Modulen oder Filtern kombiniert werden. So kann ein Objekt sowohl von der Physiksimulation wie auch von einer vorberechneten Interpolation zur gleichen Zeit beeinflusst werden.

Filter können manuell oder automatisch durch die Simulationskomponenten instanziiert werden. Dieses Vorgehen erlaubt einen **automatischen Kontrollfluss durch die Anwendungslogik**. So kann zum Beispiel automatisch eine multimodale Interpretation einer Aktion, welche ein Objekt festhält, einen Filter erzeugen, der die Kontrolle über diese Entität vollständig an die entsprechende Komponenten bindet. Die Konfliktauflösungskomponente kann in SCIVE zur Laufzeit ein- und ausgeschaltet werden. Im ausgeschalteten Zustand wird jeder Wert, der von den verschiedenen Modulen berechnet

wurde, in die zentrale Datenbasis geschrieben und mit den übrigen Komponenten synchronisiert, was dazu führt, dass immer der letzte Wert, der geschrieben wird, alle anderen überschreibt. Wird die Konfliktauflösung verwendet, werden alle Wertänderungen bis zur finalen Renderaktion verzögert. In diesem Fall werden die Werte zwischengespeichert, um zu entscheiden, in welcher Art sie gefiltert und geändert werden. Während dieser Phase kann ein Filter folgende Aktionen durchführen:

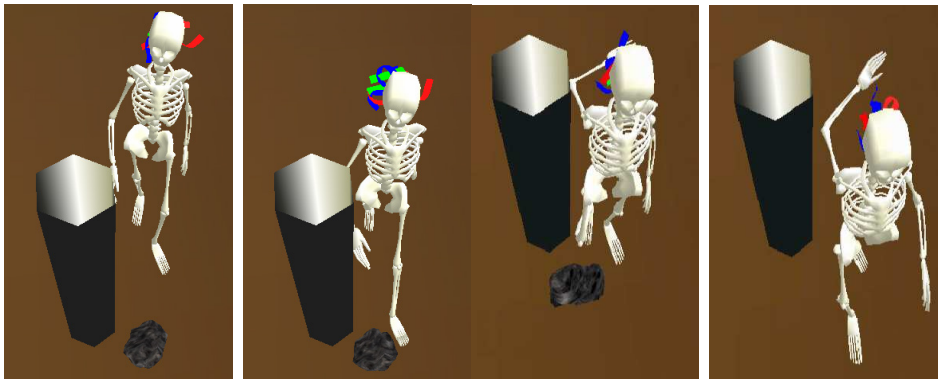
1. Einen Wert einfach **weiterleiten**
2. Einen Wert mit anderen **kombinieren**
3. Einen Wert vollständig **blockieren**

Der letzte Filter in der Kette ist mit dem Datencontainer in der zentralen Datenbasis verbunden. Nach der Auswertung der Filter und dem Propagieren der Änderungen müssen alle Module über eventuell zurückgewiesene Änderungen informiert werden. Das Beispiel des Agenten, der sich in einer virtuellen Umgebung bewegt, in Erinnerung rufend, kann dieser sowohl durch eine kinematische Skelettanimation, wie auch durch eine physikalische Simulation beeinflusst werden. Die folgenden Filter implementieren die dafür benötigten Funktionalitäten:

- **Das letzte Modul** weiterleiten.
- **Ein zufälliges Modul** weiterleiten.
- **Ein bestimmtes Modul** weiterleiten.
- **Auswahl aus einer Prioritätenliste**. Die Events aus einer Liste von priorisierten Modulen weiterleiten. Ist die Queue des in der Priorität höchsten Moduls leer, wird das nächst tiefere ausgewählt.
- **Das Physikmodul** weiterleiten. Wendet Kräfte auf die Entität für alle anderen Events an, die eine Positionsänderung hervorrufen.

- **Das Skelett-Animationsmodul** weiterleiten. Generiert eine dynamische Animation und integriert diese in mit den aktuellen Animationen für alle anderen Events mit Positionsänderungen.

Abbildung 4.4 zeigt eine aus einer Kollision zwischen dem Agenten und einem physikalisch repräsentierten Pfeiler resultierende Animation.



(a) (b) (c) (d)

Abb. 4.4: Der Agent kollidiert mit einem festen Pfeiler sowie einem Stein auf seinem Pfad. Die Kollision signalisiert die Aktivierung eines *animation blending filters*, der die entsprechenden Körperteile für die bestimmte Zeit unter Kontrolle nimmt, während die mit dem Agenten kollidierenden Objekte unter physikalischer Kontrolle bleiben [Latoschik, Fröhlich, u. Wendler 2006].

Der Benutzer kann noch weitere Filter implementieren, um beispielsweise Durchschnittswerte für eingehende Events zu berechnen. SCIVE unterstützt die Filteranwendung durch Regeln, die bestimmte Filter zu bestimmten Teilen der Szene zuordnen. Diese Teile können von unterschiedlicher Granularität sein, welche von einem Attribut einer einzelnen Entität bis zu allen in der Szene vorhandenen Attributen reicht. Die Priorisierung von Regeln stellt sicher, dass Attribute, die schon mit einer gegebenen Regel verbunden sind, nicht durch eine Regel niedrigerer Priorität überschrieben werden. Diese **priorisierte Anwendung von Regeln** stellt einen Weg dar, um eine große Menge von Filtern zuzuweisen. Dies ermöglicht es dem Benutzer, sich auf die Semantik zu konzentrieren, die er erreichen will, anstatt sich um die komplexe Definition der einzelnen Verbindungen zu sorgen.

4.2 Modulare Simulationsarchitektur

Nach der Vorstellung der technischen Grundlage im vorherigen Abschnitt wird im folgenden die zu erarbeitende Simulationsarchitektur näher ausgeführt. Dabei werden die gestellten Anforderungen, das resultierende Konzept sowie eine exemplarische Integration von bereits vorgestellten theoretischen Konzepten und Methoden aufgezeigt. Dies geschieht mit Hinblick auf die im einleitenden Kapitel 1 festgehaltene Motivation, eine **längere Persistenz von Applikationen in der Virtuellen Realität** zu erreichen. Dabei wird das definierte Ziel, eine möglichst modulare Simulationsarchitektur umzusetzen, verfolgt.

4.2.1 Anforderungen

In einer modular aufgebauten Simulationsarchitektur übernehmen die einzelnen Module dedizierte Aufgaben. Um eine umfassende virtuelle Umgebung zu simulieren, ist es notwendig, für die verschiedenen Modalitäten spezialisierte Module einzusetzen. Der Anspruch sollte dabei auf einer **Vollständigkeit der Modalitäten** basieren, die eine solche Simulation möglich machen. Im Idealfall sollten sich die Module mit geringem Aufwand gegen andere austauschen lassen, um möglichst flexibel zu bleiben. Diese Austauschbarkeit der Module trägt zu einer längeren Persistenz von VR-Applikationen bei.

In Abschnitt 1.2 wurde als Ziel für die Umsetzung einer modularen Simulationsarchitektur die Verwendung von austauschbaren Komponenten, deren Daten in eine zentrale Wissensbasis integriert werden, festgelegt. Zusätzlich wurde das Ziel definiert, bewährte Methoden aus der VR-Entwicklung bereitzustellen, um den Benutzer mit Funktionen auszustatten, die ihm aus früheren Entwicklungen bekannt sind. Diese Ziele verfolgend, sind die Anforderungen an eine möglichst modulare Architektur durch die folgenden Punkte festzuhalten:

Anforderungen an die modulare Simulationsarchitektur:

1. **Vollständigkeit** der Module, um eine umfassende virtuelle Umgebung simulieren zu können
2. **Eigenständige** parallele Berechnung der Simulationsmodule
3. **Austauschbarkeit** der verschiedenen Module
4. Integration aller Daten in eine **Wissensrepräsentation**
5. **Nahtlose Integration** bewährter Methoden in den modularen Aufbau der Simulationsarchitektur

Um diesen Anforderungen zu begegnen, wird im folgenden zunächst ein Konzept vorgestellt, das eine solche modulare Struktur umsetzt. Anschließend wird eine exemplarische Integration verschiedener Methoden gezeigt, die die Realisierung des Konzepts möglich macht.

4.2.2 Konzept

Das folgende Konzept soll die im vorigen Abschnitt festgelegten Anforderungen bezüglich einer modularen Simulationsarchitektur umsetzen. Abbildung 4.5 zeigt den konzeptuellen Aufbau eines modularen Simulationssystems, das den definierten Anforderungen gerecht wird. Es besteht aus einzelnen Modulen, die über eine Schnittstelle an den Simulationskern angekoppelt sind. Diese definiert einen Satz von Funktionen, die das Modul bereitstellen muss, um seine Daten eigenständig berechnen und seine Ergebnisse weiteren Modulen bereitstellen zu können. Soll ein Modul ausgetauscht werden, muss die eingefügte Komponente ebenfalls diesen Funktionsumfang umsetzen. Die abgebildeten Module Grafik, Physik, Audio und Interaktion setzen eine umfassende Simulation einer virtuellen Umgebung um, die verschiedene Modalitäten verwendet. Durch diese **Vollständigkeit** wird die Architektur Punkt 1 der Anforderungen gerecht. Die **eigenständige Simulation** der Module innerhalb ihres eigenen Taktes begegnet Punkt 2. Die in 3 geforderte **Austauschbarkeit** der Module wird durch die Definition einer Schnittstelle erreicht.

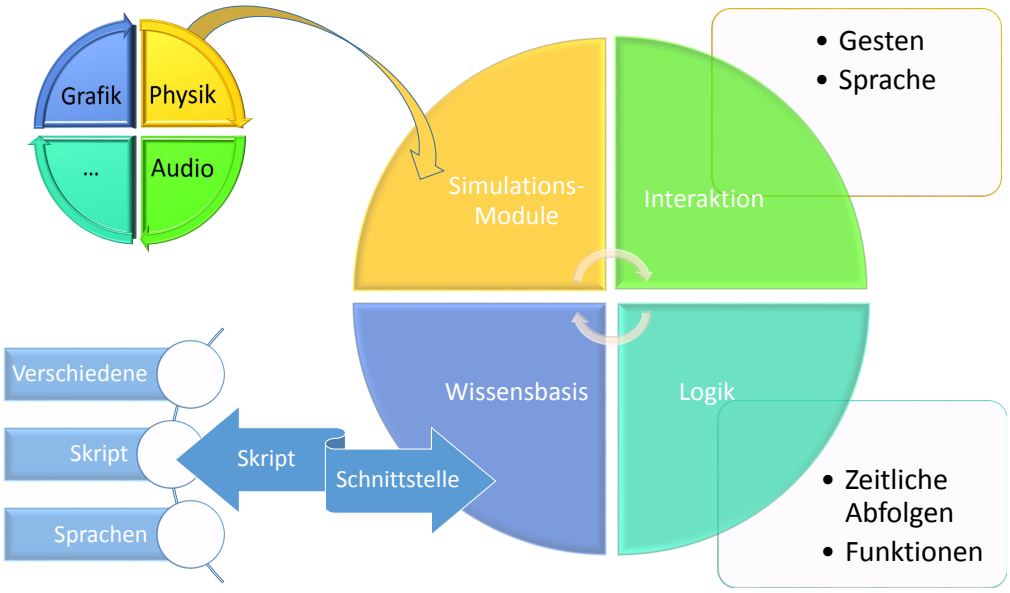


Abb. 4.5: Konzeptuelle Übersicht einer modularen Simulationsarchitektur

Des Weiteren ist in der Abbildung 4.5 eine **zentrale Wissensbasis** abgebildet, die alle, von den einzelnen Modulen gelieferten, Daten vereint. Damit wird auch der 4te Aspekt der Anforderungen umgesetzt. Die in 5 geforderte **nahtlose Integration** von bekannten Methoden ist in der Abbildung durch die Integration einer Skript-Schnittstelle illustriert. Diese setzt die Möglichkeit eines Rapid Prototypings um. Die Integration erfolgt über eine Kapselung von Operationen auf der zentralen Wissensbasis, die so zur Laufzeit der Simulation manipuliert werden kann. Das Kapseln der Funktionen sollte dabei unabhängig von einer speziellen Skript-Implementierung sein, damit auch diese Komponente ausgetauscht werden kann.

Ausgehend von dem modular aufgebauten Simulationsframework SCIVE (siehe 4.1), sollen die in diesem Konzept erarbeiteten Punkte erreicht, beziehungsweise weiter ausgebaut werden. Dazu werden im folgenden Abschnitt einige Aspekte vorgestellt, die eine Umsetzung einer solchen Architektur ermöglichen.

4.2.3 Exemplarische Integration

Eine exemplarische Integration der vorgestellten theoretischen Konzepte, im speziellen der semantischen Reflexion aus Abschnitt 3.2 und des Actor Models aus Sektion 2.3, zeigt wie eine Verbesserung der Modularität des Frameworks erreicht wird. Zunächst wird die Struktur des Actor Models betrachtet und auf den Aufbau modularer Simulationssysteme übertragen. Durch diese Übertragung wird die Schnittstelle definiert, die es braucht, um ein Modul an der Simulation teilhaben zu lassen. In einem zweiten Schritt wird auf die Anwendung des Prinzips der semantischen Reflexion in SCIVE eingegangen. Es wird gezeigt, wie die verschiedenen Komponenten ihre Daten in eine zentrale Wissensbasis integrieren. Die semantische Reflexion ermöglicht außerdem eine umfassende semantische Modellierung von weiteren Aspekten, die zu einer Simulation beitragen. Auf diese Aspekte wird weitergehend in Abschnitt 4.3 eingegangen.

4.2.3.1 Integration des Actor Models in SCIVE

Die Übertragung der Actor Model-Struktur auf der Ebene der partizipierenden Simulationsmodule und deren **zeitlicher Synchronisation** ist ein erster wichtiger Schritt, um das Modell in SCIVE zu integrieren [Fröhlich u. Latoschik 2008]. Generell können Aktoren als abgeschlossene Einheiten oder auch Primitive von Berechnungen gesehen werden.

Betrachtet man nun modulare Simulationssysteme, so fallen hier, bezogen auf die Funktionsweise von Aktoren, sehr ähnliche Strukturen auf. Jedes Modul berechnet seine eigene Domäne innerhalb seiner lokalen Begrenzungen und in seinem eigenen Takt. So berechnet beispielsweise eine physikalische Simulation Kräfte, welche auf Objekte in der Welt wirken, oder gibt generelle Umgebungsdetails wie die herrschende Gravitation vor, während sich die Grafikkomponente nur darum kümmert, wie diese Objekte anzuzeigen sind. Die Module berechnen ihre Daten also ohne Kenntnisse von den Daten der jeweils anderen zu haben.

Um solche **Simulationsmodule** nun mit einem Aktor zu vergleichen, ist es sinnvoll, sich noch einmal die Fähigkeiten anzuschauen, die ein Aktor laut dem definierten Modell haben muss. Ein Aktor oder in unserem Fall ein Modul muss:

1. **Nachrichten** von anderen Modulen **empfangen** können.
2. **Lokale Entscheidungen** treffen und **Berechnungen** durchführen können.
3. **Nachrichten** zu anderen Modulen **senden** können.
4. Neue **Module kreieren** können.

Das folgende Beispiel soll kurz verdeutlichen, dass die oben genannten grundlegenden Fähigkeiten für ein dediziertes aufgabenspezifisches Simulationsmodul ausreichen. Das Ziel ist die Durchführung einer Simulation, welche die physikalische Berechnung einer Welt sowie deren grafischer und akustischer Ausgabe mit Benutzerinteraktion beinhaltet. Grafik 4.6 zeigt drei unabhängige Module und eine **zentrale Kontrollinstanz** (SCIVE), welche die semantische Reflexionsschicht sowie den Austauschmechanismus für die Daten zwischen den einzelnen Modulen enthält. Jede Aktion in diesem Beispiel kann auf die elementaren Operationen des Actor Models reduziert werden. Die Module versenden Nachrichten untereinander, womit sie auf eingehende oder interne Events reagieren. Sie führen außerdem lokale Berechnungen aus und treffen unabhängig voneinander lokale Entscheidungen. Die Kontrollinstanz hat des Weiteren die Möglichkeit neue Module zur Laufzeit der Simulation, wie im Falle des Audio-Moduls, zu starten.

Um eine generische Komponente, beispielsweise ein bestehendes Grafik- oder Audioframework, auf Basis der Aktor-Struktur an SCIVE anzuschließen, muss es mit den grundlegenden Fähigkeiten des Modells ausgestattet werden. Zu diesem Zweck wird eine **Aktor-Schicht** entwickelt, die die benötigten Funktionen zur Verfügung stellt. Eine neue Komponente muss dann lediglich die definierte Schnittstelle erfüllen, um die Funktionen des Aktors

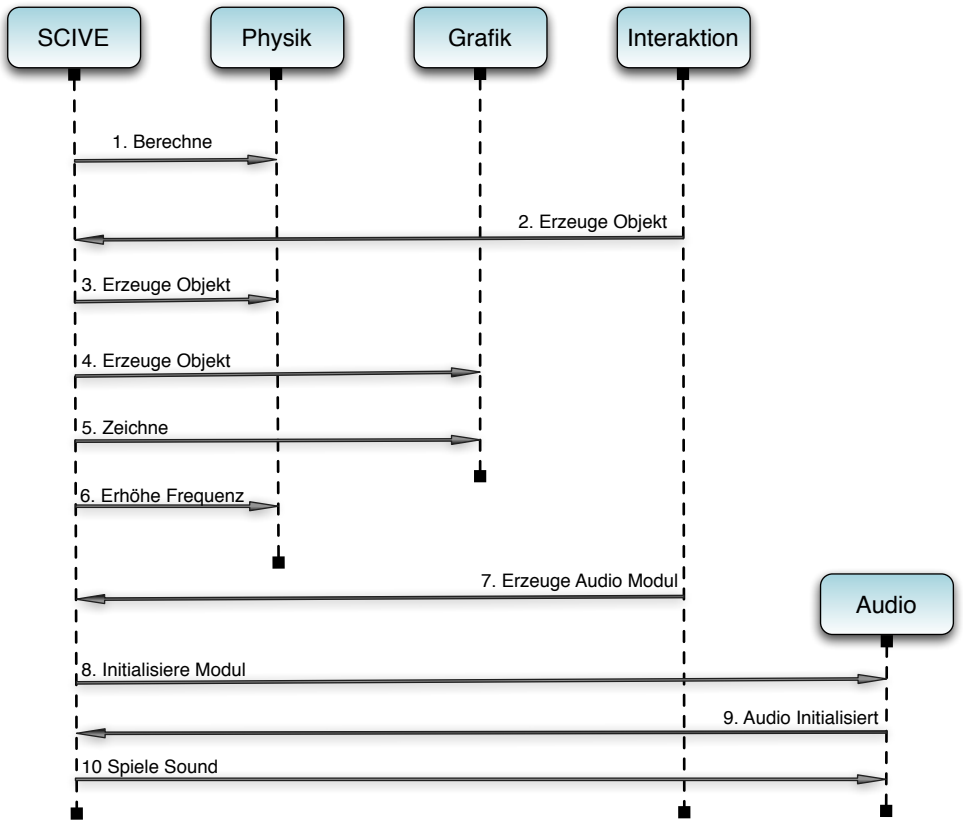


Abb. 4.6: Verschicken von Nachrichten zwischen verschiedenen Modulen in SCIVE. Die Module kommunizieren untereinander und führen Aktionen aus.

zu verwenden. **Die Simulationskomponente wird in einen Aktor gekapselt**. Abbildung 4.7 illustriert diese Vorgehensweise auf einer konzeptuellen Ebene. Eine generische Komponente, die zu der Simulation beitragen soll, wird von der Aktor-Schicht umgeben. Diese stellt die oben erwähnten grundlegenden Funktionen eines Aktors bereit. Es handelt sich hierbei um eine wiederverwendbare Komponente, die für verschiedene Module eingesetzt werden kann. Dabei erhält jedes Modul seine eigene Aktor-Schicht. Diese regelt den eigentlichen Empfang sowie das Verschicken von Nachrichten. Über den Simulationskern werden neue Aktoren initiiert. Die Berechnung von lokalen Daten wird in der Simulationskomponente selbst durchgeführt, allerdings wird zusätzlich die Möglichkeit gegeben, diese Berechnungen von außen zu starten. Für das jeweilige Modul werden nur Funktionen implementiert, die

die aktuell anliegende Nachricht abfragen, eine neue Nachricht verschicken, einen neuen Aktor über den Simulationskern erzeugen oder ihre interne Berechnungsschleife anstoßen.

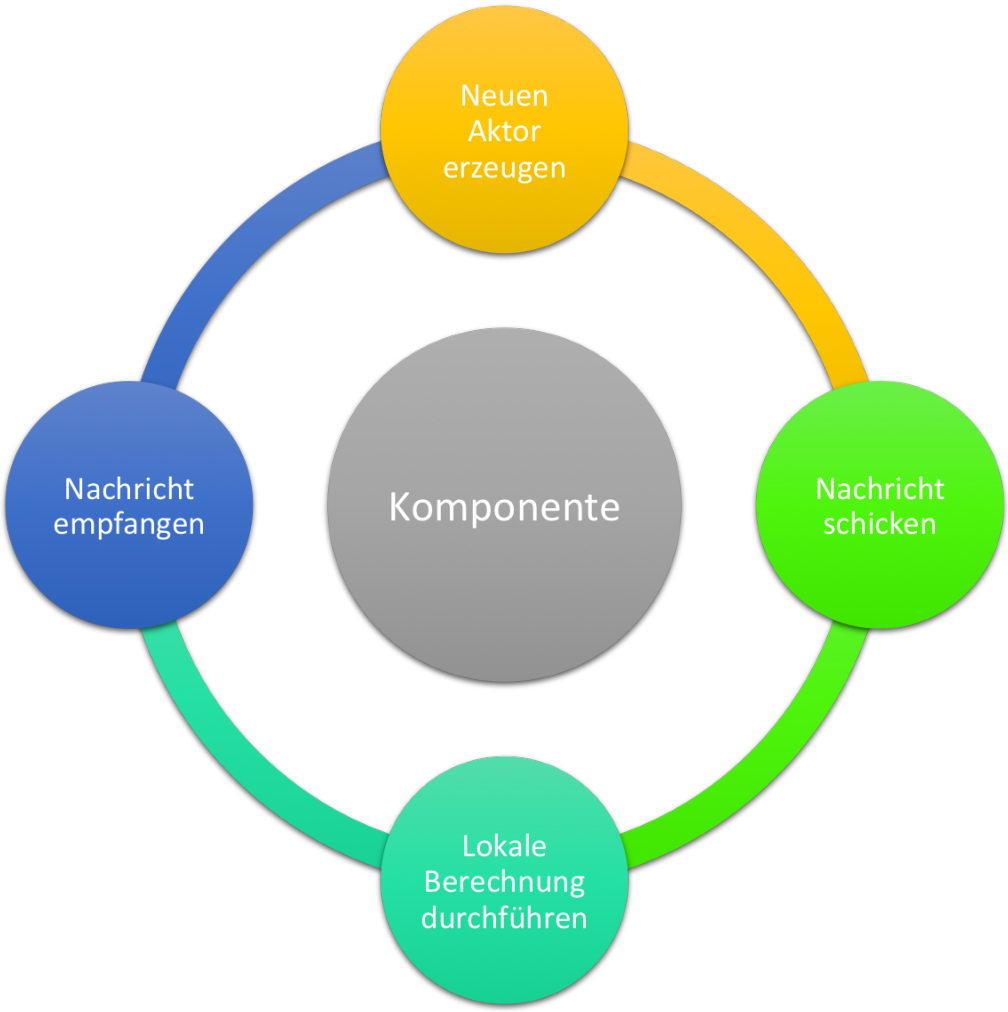


Abb. 4.7: Eine generische Simulationskomponente wird von einer Aktor-Schicht umgeben. Diese stellt die vier grundlegenden Funktionen eines Aktors bereit. Die Komponente muss nur eine Schnittstelle implementieren, die auf die Funktionen der Schicht zugreift.

Der folgende Algorithmus 1 illustriert die **Implementierung der vier grundlegenden Funktionen** eines Aktors für eine generische Simulationskomponente.

Algorithmus 1 : Die vier grundlegenden Funktionen eines Aktors werden von einem generischen Simulationsmodul implementiert

Daten : Die Schnittstelle, die eine Komponente implementieren muss, um als Aktor an einer Simulation teilzunehmen

Ergebnis : Ein Simulationsmodul, dass die vier grundlegenden Funktionen eines Aktors bereitstellt

```
1: Component c;

   // Nachrichten vom Aktor in Empfang nehmen
2: function c.GetMessage(Actor* a)
3:     Message m = a.GetActiveMessage();
4:     ProcessMessage(m);
5: end function

   // Nachrichten mit Adressat und Inhalt verschicken
6: function c.SendMessage(Actor* a)
7:     Message m = new Message(adress, content);
8:     a.SendMessage(m);
9: end function

   // Auf Anfrage interne Berechnungen anstellen
10: function c.ComputeData
11:     DoInternalComputations();
        return finished;
12: end function

   // Über SCIVE einen neuen Aktor erstellen
13: function c.SpawnActor(SCIVE* s)
14:     uniqueID id;
15:     string name;
16:     s.CreateActor(id, name);
17: end function
```

Implementiert ein Modul die geforderte Schnittstelle, kann es an eine Aktor-Schicht angeschlossen werden. Die Aktor-Schicht ist ein in C++ vorliegendes Modul, das mit einem eindeutigen Identifikator initialisiert wird, der auch im weiteren als Adresse für die Nachrichtenverarbeitung verwendet wird. Der folgende Algorithmus 2 illustriert **das Erstellen einer solchen Aktor-Schicht** und einer Simulationskomponente (Zeilen 2 bis 5). Mit der Verbindung der beiden Bestandteile wird ein neues Modul erzeugt (Zeile 6), das an das Simulationsframework SCIVE angeschlossen wird (Zeile 7). Einmal an das Framework angeschlossen, ist es über seine Adresse für andere Module und den Benutzer durch eine Skriptschnittstelle zugänglich.

Algorithmus 2 : Eine Aktor-Schicht und eine Komponente werden erzeugt und anschließend miteinander verbunden. Das verbundene Modul wird bei SCIVE registriert.

Daten : Ein Aktor und eine generische Simulationskomponente
Ergebnis : Ein Simulationsmodul, das mit einer Aktorschicht verbunden und mit seiner Adresse registriert wurde

```
1: function CreateNewActorModule
       // Bestandteile initialisieren
2:     uniqueID id;
3:     string name;
4:     Component c;
5:     Actor a = new Actor(id, name);
       // Komponente und Aktor verbinden
6:     Module m = c.ConnectToActor(a);
       // Neues Modul bei SCIVE registrieren
7:     SCIVE.Register(m);
       return m;
8: end function
```

4.2.3.2 Semantische Reflexion in SCIVE

Semantische Reflexion soll in SCIVE einen dynamischen und ontologiebasierten Zugriff auf alle Module und Entitäten zur Laufzeit einer Simulation bereitstellen. Als **Ontologie** wird hier eine sprachlich gefasste und strukturell geordnete Darstellung einer Menge von Begrifflichkeiten verstanden. Zwischen diesen bestehen Verbindungen (auch **Relationen** genannt), die eine Bedeutung modellieren. Es wird also Wissen in formaler Struktur vorgehalten, dass zwischen Komponenten ausgetauscht werden kann. Ein ontologiebasierter Zugriff bezeichnet demnach den Zugriff auf einer wissensbasierten sprachlich gefassten Ebene. Um diesen für die gesamte Simulation zu ermöglichen, muss er auf verschiedenen Ebenen realisiert werden (vergleiche Abbildung 4.8).

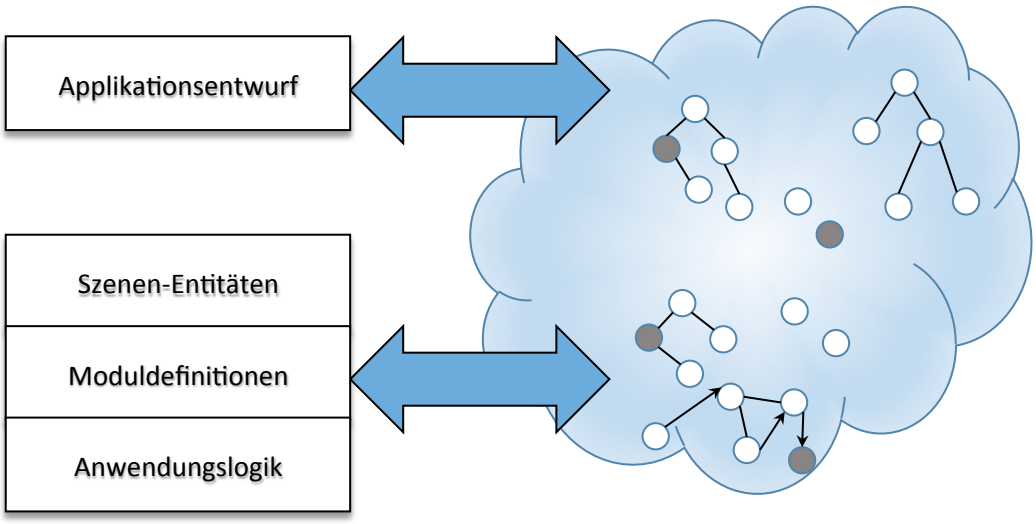


Abb. 4.8: Semantische Reflexion bildet Daten und Objektrepräsentationen verschiedener Ebenen auf eine gemeinsame Wissensrepräsentation ab. Übernommen von Latoschik u. Fröhlich [2007b]

Zunächst muss der Simulationskern selber abgebildet werden, d.h. welche Module sind beteiligt und wie wird der Datenfluss zwischen diesen geregelt? Dies wird unter der Modellierung der Anwendungslogik zusammengefasst. Des Weiteren müssen die Spezifikationen der einzelnen Module repräsentiert

werden. Es sollten hierbei verschiedene Formen möglich sein, z.B. ein Szenengraph oder auch einfache einzelne Entitäten innerhalb der physikalischen Simulation. Außerdem müssen in einem dritten Schritt diejenigen Entitäten abgebildet werden, aus denen sich die dargestellte Szene zusammensetzt. Diese machen die möglichst realistische Erfahrung für den Benutzer aus.

In SCIVE wird die semantische Reflexion durch ein Netz von Definitionen verbundener Entitäten implementiert. Es wird eine **eigenständige Schicht für die Wissensrepräsentation** bereitgestellt, die die verteilten Weltrepräsentationen mit einander verbindet. Jedes Applikationsobjekt und jede Entität der verschiedenen Schichten aus Abbildung 4.8 wird in der Wissensrepräsentation gespiegelt.

Initial werden alle Module mit einer *Wrapper*[1]-Schnittstelle ausgestattet. Diese stellt die benötigten Verbindungen zu der Logik des Simulationskerns her. Darüber hinaus verbindet sie außerdem die speziellen Modulobjekte mit ihren Gegenstücken in der semantischen Reflexionsschicht. Die Wissensrepräsentationsschicht verbindet diese Gegenstücke – Knoten eines semantischen Netzwerkes – mit den dazu passenden Objektbeschreibungen anderer Module. Sogenannte *Slots* bieten die Möglichkeit, zusätzliche Informationen innerhalb der Knoten zu speichern. Die Knoten stellen eine logische Schnittstelle zu konkreten Objektrepräsentationen bereit. Die auf diese Art und Weise durchgeführte semantische Reflexion stellt einen wissensbasierten Zugang zu verteilten Datenrepräsentationen bereit, welcher über die aus der objektorientierten Programmierung bekannten Vorgehensweise hinaus geht.

Die Vorteile einer mittels semantischer Reflexion implementierten Anwendung sollen (wie schon in Abschnitt 4.1) am Beispiel der Kommunikation mit einem virtuellen Agenten betrachtet werden. Dafür muss zum einen eine Benutzerinteraktion, und zum anderen eine Perzeption des Agenten realisiert werden.

[1] **Wrapper**: von engl. *Umschlag* oder *Verpackung*. Bezeichnet ein Stück Software, das andere Software umhüllt, um Zugriff auf verschiedene Funktionen zu kapseln

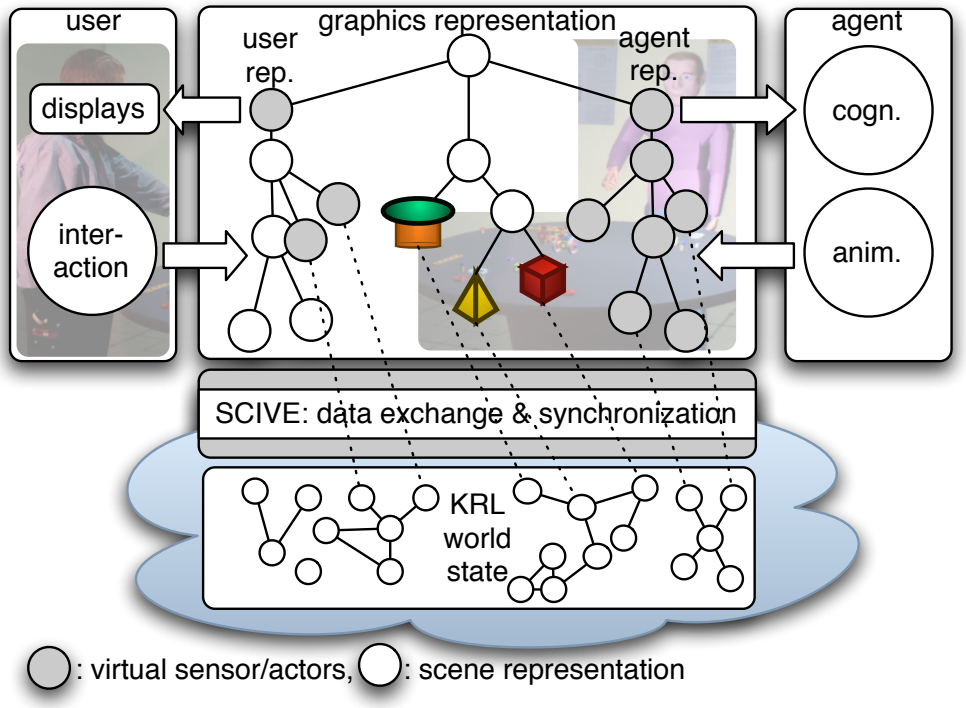


Abb. 4.9: Integration der Benutzerinteraktion und Perzeption des Agenten in die Grafikkomponente. Abbildung aus [Latoschik u. Fröhlich 2007b].

Abbildung 4.9 illustriert eine eingebettete Implementierung innerhalb der Grafikkomponente. Sowohl der Benutzer, wie auch der virtuelle Agent werden durch spezifische semantic entities im Grafikmodul repräsentiert (in der Grafik durch gepunktete Linien gekennzeichnet). Die **semantischen Repräsentationen** werden direkt im Szenengraph an ihren korrespondierenden grafischen Objekten platziert. Dies resultiert in einer starken Abhängigkeit zu der benutzten Grafikengine und kompliziert somit die in Kapitel 1 geforderte Wiederverwertbarkeit und Portabilität.

Abbildung 4.10 zeigt hingegen einen alternativen von SCIVE angebotenen Ansatz. Die dargestellte Architektur zeigt eine Abbildung der Anwendungslogik, welche durch den Zugriff auf die Wissensrepräsentationsschicht realisiert wird, ohne dass eine direkte Abhängigkeit zu einem bestimmten Mo-

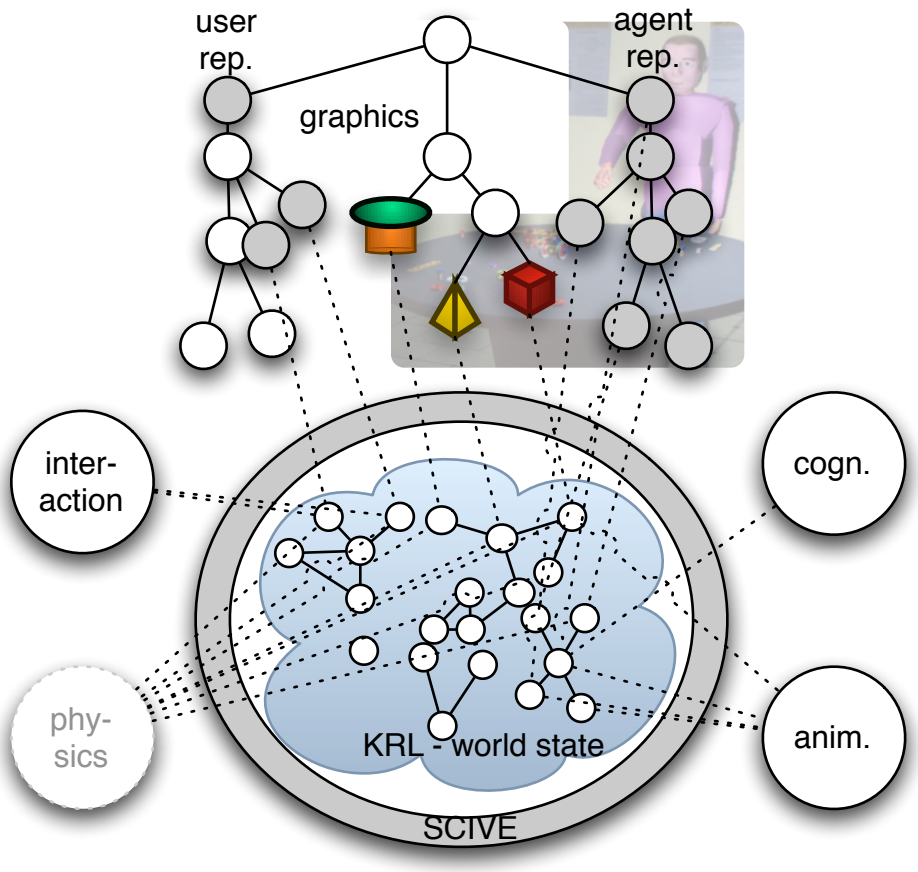


Abb. 4.10: Die Benutzerinteraktion und Perzeption des Agenten werden in der Wissensrepräsentationsschicht abgebildet. Abbildung aus [Latoschik u. Fröhlich 2007b]

dul besteht. Dadurch wird ein **erhöhter Abstraktionsgrad** im Vergleich zu der in Abbildung 4.9 gezeigten Architektur erreicht. Auch wenn die Wissensrepräsentation in dieser Abbildung eine zentrale Stellung einnimmt, sind die Module untereinander durch ihre Schnittstellen zu SCIVE verbunden, ohne von einander abhängig zu sein. Dies schafft die Möglichkeit, Module mit minimalem Aufwand auszutauschen. Da die Wissensrepräsentation die komplette Applikation reflektiert, steht dem Anwendungsentwickler auch eine feingranulare Kontrolle über den herrschenden Datenfluss zur Verfügung. Die in Abschnitt 4.1.1 beschriebenen Mechanismen des Kontrollflusses regeln

dabei, wie die errechneten Daten in die gemeinsame Wissensrepräsentation integriert werden (nähere Informationen hierzu finden sich auch in [Latoschik, Fröhlich, u. Wendler 2006]). Der Entwickler kann den Datenfluss durch die Definition von Filtern beeinflussen, welche regeln, in weit die einzelnen Module ihre Daten miteinander verrechnen. Ein **konsistenter Weltzustand** wird dadurch erreicht, dass alle Wertänderungen durch die verschiedenen Komponenten bis zur finalen Darstellung eines Frames verzögert werden. Wird eine Komponente ausgetauscht, startet die neu hinzugefügte mit dem zum Zeitpunkt des Einstiegs herrschenden Weltzustand.

Um die Vorteile von semantischer Reflexion weitergehend zu illustrieren, betrachten wir noch einmal das Beispiel des virtuellen Agenten, im Speziellen seine visuelle Perzeption sowie die Abbildung 4.10. Eine Möglichkeit, eine solche perzeptive Komponente für einen Agenten zu entwerfen, ist die Entwicklung eines **Sensors**, der automatisch alle Objekte untersucht, die in seiner Richtung und Reichweite liegen. Da dieses in jedem dargestellten Frame durchgeführt werden muss, wird die Umsetzung oft innerhalb der Grafikkomponente implementiert. Dies ist außerdem sinnvoll, weil der Sensor auf diese Art und Weise Zugriff auf die ihn umgebenen grafischen Objekte erhält. Um entscheiden zu können, welche Objekte wahrgenommen oder manipuliert werden können, muss eine gewisse Parametrisierung vorgenommen werden. Diese kann realisiert werden, indem mittels der zu Grunde liegenden Programmiersprache Objekte eines bestimmten Typs entwickelt werden, die der Sensor wahrnehmen kann. Ein solches Vorgehen wäre allerdings sehr umständlich, da jede Änderung ein Kompilieren und Neustarten der Anwendung zur Folge hätte. Mittels semantischer Reflexion lässt sich ein komfortableres Vorgehen erreichen, da die semantische Wissensrepräsentationsschicht zur Laufzeit gelesen und geändert werden kann.

Durch den Einsatz semantischer Informationen in einer einheitlichen Wissensrepräsentation wird es ermöglicht, eine Selektion der wahrnehmbaren Objekte durchzuführen. So kann zum Beispiel nur nach Objekten gesucht werden, welche den Eintrag `is-agent-observable` enthalten. Somit ist der

View Sensor in der Lage, das semantische Netz in jedem Simulationsschritt nach genau den Knoten abzusuchen, die dieses spezielle Attribut verwenden, da er „weiß“, wonach er suchen muss. Wenn der Sensor seine gesuchten Objekte gefunden hat, ermöglicht ihm die semantische Reflexion einen Zugriff auf die Positionsdaten der Objekte in der Grafikkomponente und er kann somit eine räumliche Zuordnung aufbauen. Das Durchsuchen des Netzwerks nach genau diesen Informationen wird durch den Einsatz der semantischen Traverser erledigt (siehe auch Abschnitt 4.3.3). Auf technischer Ebene ist ein solcher View-Sensor also eine Komponente des virtuellen Agenten, der in der **einheitlichen Wissensbasis reflektiert** wird. Ausgehend von dem Sensor wird das Netz nach den benötigten Informationen durchsucht, die wahrgenommen werden können.

Die hier beschriebene semantische Reflexion ermöglicht die Modellierung von allen Aspekten einer VR-Simulation, die im folgenden Abschnitt beschrieben wird.

4.3 Semantische Modellierung

In den vorherigen Abschnitten wurde die Grundlage für ein modulares Simulationsframework gelegt. Diese Modularität setzt auch eine **Integration der Berechnungsergebnisse** in einer gemeinsamen Wissensrepräsentation voraus. Dies soll in einer möglichst abstrakten Ebene realisiert werden, um weiterhin unabhängig von speziellen Modulen zu bleiben. Die semantische Modellierung der Daten stellt den zweiten wichtigen Aspekt der vorliegenden Arbeit dar. Im Folgenden wird gezeigt, wie sie durch die Modellierung verschiedener Aspekte zu der erarbeiteten Simulationsarchitektur beiträgt. Dazu werden zunächst wieder die Anforderungen festgehalten, bevor eine konzeptuelle Umsetzung sowie eine exemplarische Integration vorgestellt wird.

4.3.1 Anforderungen

Mit der semantischen Modellierung virtueller Umgebungen soll eine höhere Abstraktion der Simulationsinhalte und der Anwendungslogik erreicht werden. Diese Abstraktion soll unabhängig von speziell implementierten Modulen sein. Damit wird das in Kapitel 1 gesetzte Ziel des längeren Fortbestands von VR-Applikationen verfolgt. Der Begriff der semantischen Modellierung wurde in der Einleitung von Kapitel 3 mit Definition 5 festgehalten. Bei der hier vorgeschlagenen semantischen Modellierung werden getrennte Bereiche betrachtet, die einerseits die Inhalte modellieren, womit die in der Definition gestellte Frage nach dem Subjekt beantwortet wird. Die Fragen nach dem Zeitpunkt und dem Zweck fallen in den Bereich der Anwendungslogik. Die Modellierung dieses Bereiches sieht vor, zeitliche Abfolgen, die Definition von Funktionen und die **Integration semantischer Traverser** abzubilden.

Anforderungen an die semantische Modellierung:

1. Es soll eine **umfassende** Modellierung virtueller Umgebungen ermöglicht werden.
2. Die gesamte Modellierung soll in einer **einheitlichen** Repräsentation vorgehalten werden.
3. Die Repräsentation des Wissens soll **unabhängig** von spezifischen Modulen sein.
4. Es soll eine Möglichkeit für die Modellierung der **Anwendungslogik** geschaffen werden. Die Logik einer Anwendung umfasst, wie in Abschnitt 1.1 festgehalten, die Zusammenhänge nach denen diese funktioniert. Dazu gehören beispielsweise zeitliche Abfolgen oder die Definition von konkreten Funktionen.

Die hier aufgelisteten Punkte werden im folgenden Abschnitt anhand eines Konzeptes ausgearbeitet. Dabei wird eine Art der semantischen Modellierung von Wissen aufgezeigt, die den Anforderungen genügt, ohne sich auf eine spezifische Umsetzung der Repräsentation und Modellierung festzulegen.

4.3.2 Konzept

Bei der semantischen Modellierung von Szenen mittels einer **einheitlichen Wissensrepräsentation** werden alle in der Szene enthaltenen Entitäten in der vorliegenden Struktur abgebildet. Die speziellen Repräsentationen werden dabei von einer allgemeinen abgeleitet. Alle in der Szene sichtbaren Objekte verwenden in der eingesetzten Struktur, neben ihren generellen Abbildungen, eine Repräsentation in der Grafik-Domäne. Diese Darstellungen werden mittels definierter Relationen miteinander verbunden.

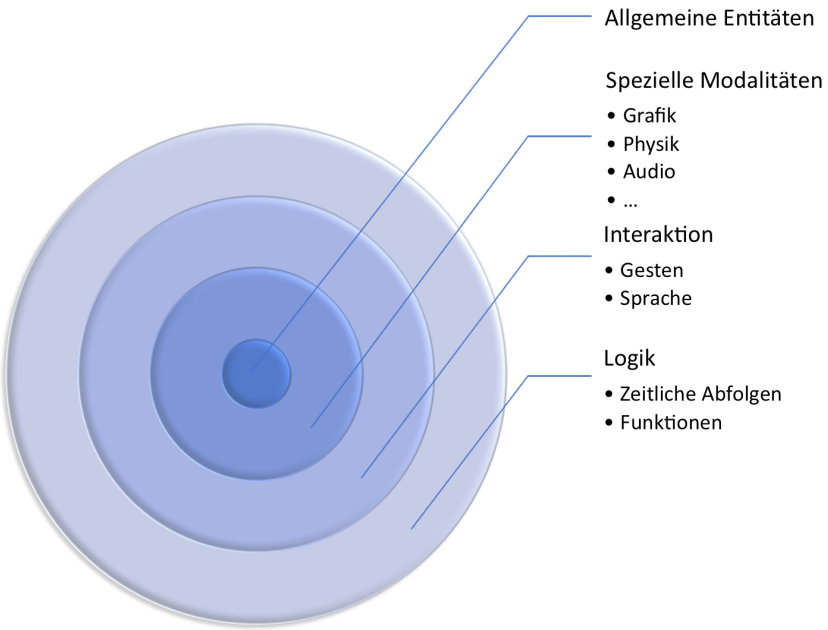


Abb. 4.11: Konzeptuelle Darstellung einer einheitlichen Wissensrepräsentation. Die verschiedenen Repräsentationen sind in getrennte Bereiche gegliedert. Zentral ist die Definition allgemeiner Entitäten dargestellt. Von diesen abgeleitet werden in den weiteren Schichten spezielle Repräsentationen aufgebaut.

Die konzeptuelle Umsetzung der semantischen Modellierung ist in Abbildung 4.11 veranschaulicht. Sie illustriert die Wissensrepräsentation des modularen Simulationssystems. Wie gefordert werden verschiedene Aspekte der Simulation abgebildet. Im Zentrum stehen allgemeine Definitionen der verschiedenen Entitäten, die zur Simulation der Szene beitragen. Aufbauend auf diesen werden die weiteren Schichten modelliert. Es werden spezielle Repräsentationen für verschiedene Modalitäten, beispielsweise Grafik und Audio, abgeleitet. Darüber hinaus werden Informationen bezüglich der Benutzerinteraktion modelliert. Eine weitere Schicht steht für die Anwendungslogik zur Verfügung. Durch die Aufteilung in getrennte Bereiche, die nach individuellen Bedürfnissen gestaltet werden können, soll eine umfassende Modellierung nach Anforderung Nummer 1 ermöglicht werden. Die **Trennung der Sektionen** erfolgt, um die Repräsentation für den Entwickler übersichtlicher zu gestalten. Außerdem können auf die Art einzelne Bereiche der Anwendung bei Bedarf an- und ausgeschaltet werden.

Obwohl die Repräsentation der gesamten Simulation in unterschiedliche Schichten aufgeteilt ist, basiert sie auf derselben Struktur. Hierfür wird an dieser Stelle ein semantisches Netz vorgeschlagen. Diese eignen sich, wie schon in Abschnitt 3.3 dargestellt, sehr gut zur hierarchisch strukturierten Form der Wissensrepräsentation. Die verschiedenen Schichten der Repräsentation werden durch unterschiedliche Bereiche im Netz abgebildet, die bei Bedarf mittels Relationen mit anderen Bereichen verknüpft werden können. Die Integration des gesamten Wissens in ein semantisches Netz begegnet Anforderung Nummer 2.

Durch die abstrakte Art der Wissensmodellierung soll die **Unabhängigkeit** von spezifischen Simulationsmodulen maximiert werden, die in Anforderung Nummer 3 gefordert wurde. Die Wissensmodellierung findet nicht direkt in den jeweiligen Formaten der verschiedenen Komponenten statt, sondern wird ausgelagert. Auf diese Weise müssen lediglich Schnittstellen definiert werden, die festlegen, wie ein Modul Daten aus dem Netz lesen bzw. seine errechneten Ergebnisse wieder in dieses integrieren kann. Die oben erwähnte

Repräsentation der **Anwendungslogik** wird ebenfalls in dem selben Netz vorgenommen. Dieser Punkt begegnet Anforderung Nummer 4. Durch diese Art der Integration wird auch die Logik einer Simulation abstrahiert. Abgebildet werden im Bereich der Logik zeitliche Abfolgen sowie Funktionen, die von der Anwendung bereitgestellt werden. In den Bereich der Logik fallen auch die Definitionen von semantischen Traversern, die in Abschnitt 3.4 eingeführt wurden.

Die folgenden Abschnitte zeigen, wie die semantische Modellierung virtueller Umgebungen auf Basis von SCIVE durchgeführt werden kann. Diese Ausarbeitung bedient sich dabei dem in Abschnitt 4.2.3 integrierten Prinzip der semantischen Reflexion. Es wird die Modellierung verschiedener Aspekte illustriert, die später in Kapitel 5 in Form konkreter semantischer Beschreibungen exemplarisch umgesetzt werden.

4.3.3 Exemplarische Integration

Das in Abschnitt 3.3 beschriebene funktional erweiterbare semantische Netzwerk wird in SCIVE als zentrale Wissensbasis für alle Aspekte einer Simulation verwendet – angefangen bei den an der Simulation beteiligten Modulen bis hin zu den zahlreichen Entitäten, die die simulierte Anzeige gestalten. Als Beschreibungssprache wird die *Semantic Net Interchange Language* (kurz SNIL) verwendet. Hierbei handelt es sich um ein von Latoschik u. Fröhlich [2007b] beschriebenes deklaratives XML-basiertes Format, das es dem Entwickler erlaubt, Inhalte auf textueller Basis zu definieren. Dabei werden Knoten und Relationen in XML-Umgebungen geschachtelt und ihre Beziehung untereinander definiert. Exemplarische Definitionen unterschiedlicher Aspekte auf dieser Beschreibungsebene finden sich in Abschnitt 5.2. Auf einer hohen Abstraktionsebene werden mittels Domänen die Simulationskomponenten integriert. Diese stellen semantisch getrennte Bereiche im Netz bereit, in denen die einzelnen Bestandteile der Simulation mit ihren spezialisierten Repräsentation angesiedelt sind. Abgesehen von dieser Einteilung und der Abbildung der Entitäten bietet das entwickelte Framework

auch die Möglichkeit, **abstraktere Konzepte** auf dem semantischen Netz abzubilden und zur Laufzeit zu verarbeiten. Im Folgenden wird die exemplarische Integration von beschrieben Konzepten, wie zum Beispiel Allens Intervallalgebra, beschrieben. Dabei wird gezeigt, wie verschiedene Aspekte in der erarbeiteten Architektur modelliert werden können.

4.3.3.1 Virtuelle Inhalte

Auf Basis des semantischen Netzwerkes werden virtuelle Inhalte, die der Benutzer erleben kann, entwickelt. Dazu werden die Objekte wie oben erwähnt in Domänen eingeteilt. Immer abgeleitet von einer generellen Entität werden dabei die spezifischen Entitäten mittels definierter Relationen eingefügt. Abbildung 4.12 illustriert die konzeptuelle Modellierung virtueller Inhalte im einheitlichen semantischen Netz. Abgebildet ist in der Mitte ein Bereich, der die allgemeinen Entitäten enthält. Von diesem ausgehend werden, wie angedeutet, die speziellen Objekte abgeleitet.

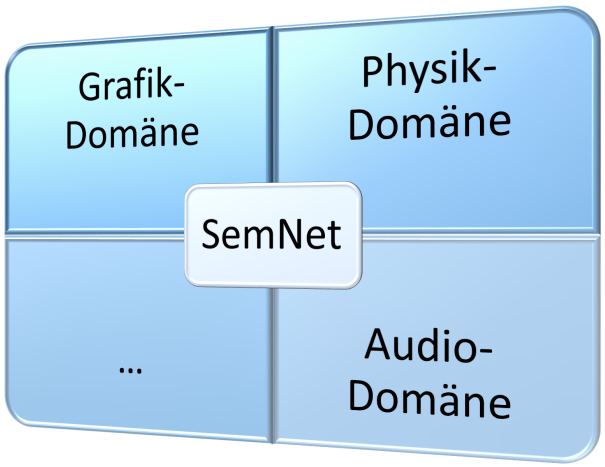


Abb. 4.12: Konzeptuelle Modellierung virtueller Inhalte im einheitlichen semantischen Netz. Die speziellen Repräsentationen werden dabei in Domänen eingeteilt.

Die Verfahrensweise bei der Modellierung virtueller Objekte wird in Abbildung 4.13 illustriert. Ausgehend von dem allgemeinsten Konzept `Virtual Object` wird ein `Virtual Ball` mit einer Instanz `Ball-1` abgeleitet. Dieses Objekt hat von sich aus noch keine Informationen über eine etwaige grafische oder physikalische Repräsentation. Diese werden erst durch die zusätzlichen Knoten in Verbindung mit den entsprechenden Relationen hinzugefügt. Der Instanz `Ball-1` werden auf diese Art und Weise grafische, akustische, physikalische und haptische Informationen zugewiesen. Diese allgemeine Art der Modellierung stellt die Grundlage einer darzustellenden virtuellen Umgebung dar.

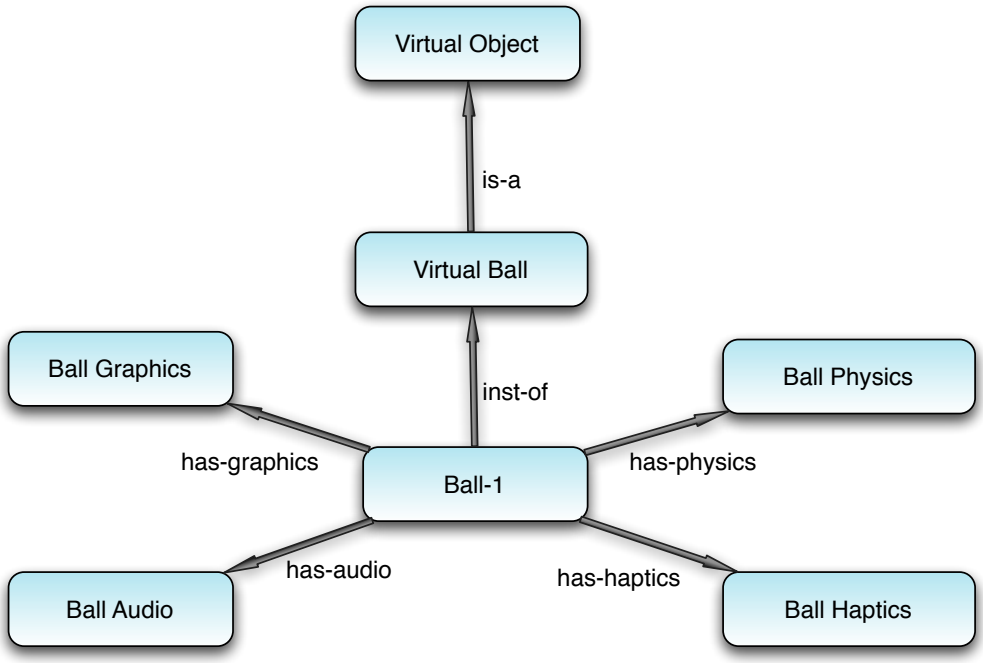


Abb. 4.13: Konzeptuelle Modellierung virtueller Objekte. Die speziellen Repräsentationen (Grafik, Physik, Audio und Haptik) werden von einem generellen Objekt abgeleitet.

4.3.3.2 Semantische Traverser

Die semantischen Traverser, die in Abschnitt 3.4 besprochen wurden, können ebenfalls in einer semantischen Art und Weise modelliert werden.

Dem Prinzip der **semantischen Reflexion** folgend, wird jeder Traverser auch als Teil des semantischen Netzes, entweder als einzelner, oder durch einen Zusammenschluss mehrerer Knoten, definiert. Dies bringt einige Vorteile mit sich. Beispielsweise erhalten die Traverser durch die Integration in die Wissensbasis Zugriff auf alle semantisch reflektierten Entitäten und modulspezifischen Daten, einschließlich anderer Traverser und ihrer Resultate. Dadurch wird die Schnittstelle zwischen Traversern und deren Ergebnissen vereinfacht. Traverser erhalten auf die gleich Art und Weise Zugriff auf die Resultate anderer Traverser wie auf die anderen Daten im semantischen Netzwerk, es ist also keine spezielle Implementierung für den Zugriff nötig. Es ist so möglich eine Art **Meta-Traverser** zu entwickeln, der die semantische Repräsentation ausschließlich nach Traversern durchsucht und ihre Ergebnisse zentralisiert sammelt. Semantische Traverser erhalten beim Start einer Aktion in der Regel einen Startknoten, von dem aus sie operieren. Wird kein Startknoten angegeben, gibt es alternativ die Möglichkeit, eine Domäne anzugeben, innerhalb derer der Traverser arbeiten soll.

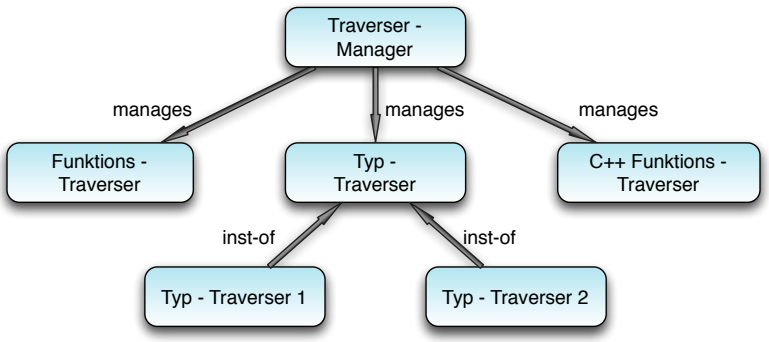


Abb. 4.14: Schematische Abbildung des Traverser-Managers, der die verschiedenen auf dem Netz operierenden Traverser koordiniert. Abbildung nachgezeichnet aus [Peuser u. Latoschik 2008].

Auf dem semantischen Netz können verschiedene Traverser parallel operieren. Zu diesen zählen zum Beispiel Traverser für Funktionsaufrufe oder Traverser für verschiedene Objekttypen. Um die Vielzahl der auf dem Netz arbeitenden Traverser verarbeiten zu können, existiert ein Manager, der Zugriff auf die verschiedenen Objekte bietet. Abbildung 4.14 zeigt den Manager, der den Zugriff auf die verschiedenen aktiven Traverser regelt.

Abbildung 4.15 zeigt die konzeptuelle semantische Modellierung eines semantischen Traversers in der gemeinsamen Wissensbasis. Abgeleitet von dem generellen Konzept `Traverser` wird das spezielle Konzept des Funktions-Traversers (`Function Traverser`) definiert. Eine spezielle Instanz davon ist der `C++ Traverser`. Dieser enthält ein Ergebnis sowie einen Startknoten.

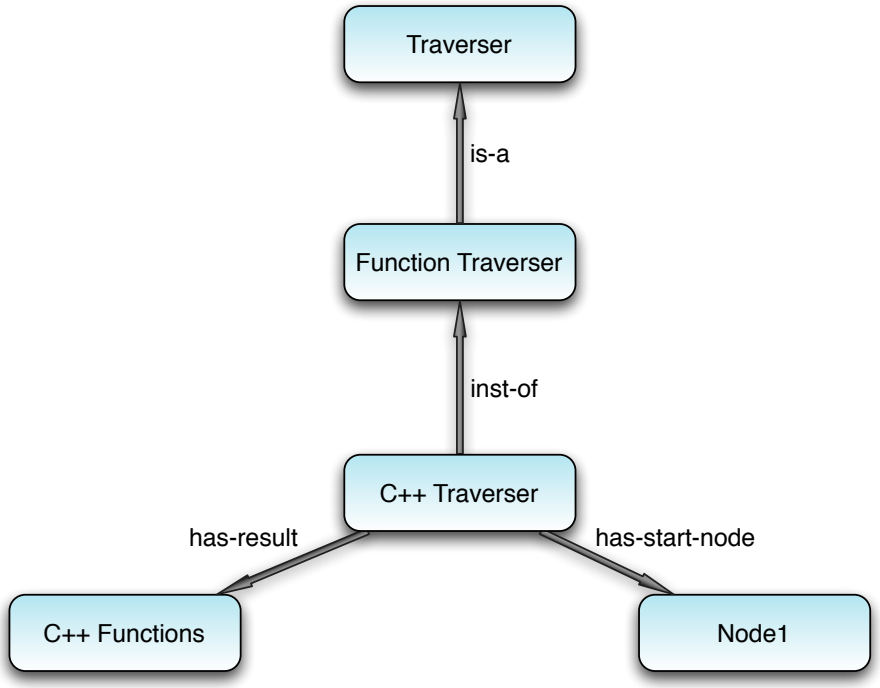


Abb. 4.15: Konzeptuelle Modellierung eines semantischen Traversers im einheitlichen semantischen Netz.

4.3.3.3 Interaktiv generierte Objekte

Neben den klassischen in der VR verwendeten Modalitäten Grafik, Audio und Physik ist auch die Modellierung von Eingabe-Modalitäten, zum Beispiel eine Verarbeitung von Gesten [Fröhlich u. a. 2009a], möglich.

Die Verarbeitung von **ikonischen Gesten** basiert auf dem Konzept der Imagistic Description Trees [Sowa 2006]. Dieses Konzept lässt sich gut als Graph darstellen und somit einfach auf ein semantisches Netz übertragen. Die Verarbeitung der im semantischen Netz abgebildeten Geste kann dann mittels definierter Relationen und Traversern erfolgen. Die gewünschte Struktur der Geste kann durch die Modellierung im Netz auch zur Laufzeit verändert werden, um etwaige Anpassungen beziehungsweise Optimierungen für den Benutzer durchführen zu können. Eine beispielhafte Implementierung einer solchen Geste wird im weiteren Verlauf der Arbeit in Abschnitt 5.2.3 gezeigt.

Auch die **parallele Verarbeitung von Gesten und Sprache** lässt sich auf dem semantischen Netz abbilden. Die Generierung von neuen Objekten mit sprachbegleiteter ikonischer Gestik ist in Abbildung 4.16 illustriert. Wie in der Abbildung zu sehen ist, werden zwei **lineare Bewegungssegmente** erkannt und verarbeitet. Zwischen diesen Segmenten wird ein Winkel errechnet, der die Krümmung des Rohres bestimmt. Die Länge der beiden Segmente legt die Länge des Rohres fest. Diese Informationen werden dem virtuellen Bauteil – in Form eines IDT – als semantische Information zugefügt, damit es später durch eine ikonische Geste referenziert werden kann. Bei der Referenzierung wird ein neuer IDT basierend auf der Geste erzeugt, welches mit dem des Bauteiles verglichen wird. Existieren mehrere solcher Bauteile, wird dasjenige ausgewählt, das bei dem Vergleich der IDTs den besten Wert erzielt hat. Um eine solche Realisierung mittels des semantischen Netzes erreichen zu können, müssen einige Informationen modelliert werden [Biermann u. a. 2007].

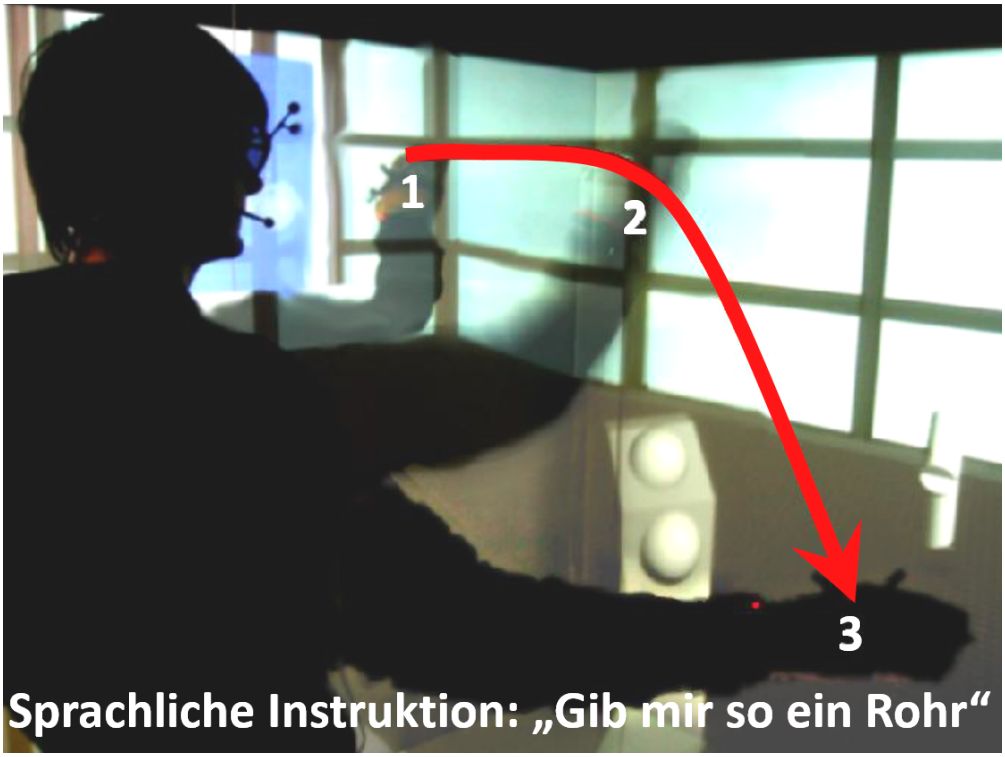


Abb. 4.16: Verwendung einer ikonischen Geste in einer virtuellen Welt mit begleitender verbaler Instruktion. Abbildung übersetzt [Fröhlich u. a. 2009a]

Abbildung 4.17 zeigt einen Überblick der benötigten Informationen, um ein Bauteil mittels ikonischer Gesten, kombiniert mit sprachlichen Instruktionen, generieren zu können. Das Bauteil „Elbow“ ist in der Mitte dargestellt. Spezielle Relationen verbinden es mit den spezialisierten Informationen. Die abgebildete Grafik zeigt eine Vererbungshierarchie. Das Objekt `Template Elbow` ist von der Oberklasse `Parametric Part` abgeleitet. Beide Konzepte sind mit einer **lexikalischen Repräsentation** verbunden, mittels welcher sie sprachlich referenziert werden können. Das eigentliche virtuelle Bauteil ist eine Instanz des `Elbow Templates` und wird mit konkreten Werten gefüllt, beziehungsweise verbunden. Während es die lexikalischen Repräsentationen der Oberklassen erbt, wird ihm noch eine weitere konkretere zugewiesen. Das Teil erhält außerdem weitere Zusatzinformationen, wie etwa, dass es biegsam ist, oder dass die grundlegende geometrische Form zylindrisch ist. Des Weiteren werden dem gebogenen Rohr zwei Verbindungsstellen zugewiesen, mit

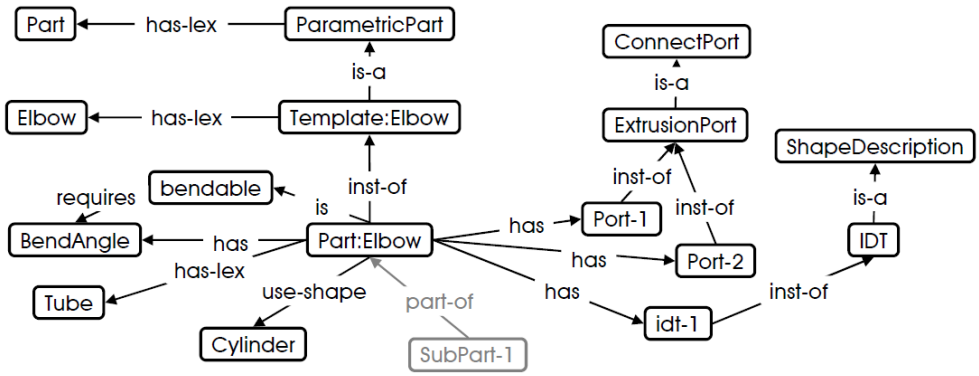


Abb. 4.17: Semantische Informationen zur interaktiven multimodalen Generierung von Bauteilen. Abbildung aus [Biermann, Fröhlich, Latoschik, u. Wachsmuth 2007].

denen es mit anderen Bauteilen zu größeren Baugruppen zusammen gefügt werden kann.

Im Bild 4.17 ist rechts unten die **ikonische Repräsentation** in Form eines IDTs dargestellt, die an dieser Stelle vereinfacht in einem Knoten abgebildet und seinerseits wiederum von einem generellen Konzept abgeleitet ist. Mit Hilfe dieser ikonischen Information kann das Bauteil auch durch eine formbeschreibende Geste referenziert werden. Hierbei wird der IDT der Geste, mit dem in dem Bauteil gespeicherten, auf eine Übereinstimmung verglichen. Die semantische Modellierung ikonisch dargestellter Formen und Gesten, die unter anderem für die virtuelle Konstruktion genutzt werden können (siehe auch Sektion 6.2), wird in Abschnitt 5.2.3 konkretisiert und exemplarisch im verwendeten semantischen Netz ausgeführt.

4.3.3.4 Funktionen

Das funktional erweiterbare semantische Netzwerk bietet in Kombination mit speziellen Traversern und der Aktor-Struktur ebenfalls die Möglichkeit, Funktionsaufrufe nahezu beliebiger Programmiersprachen auf einer semantischen Ebene zu modellieren. Diese Funktionsaufrufe implementieren Aktionen, welche einen Nachrichtenaustausch zwischen verschiedenen Aktor-

Modulen abbilden, oder Aufrufe innerhalb der Simulationsschleifen einzelner Module definieren. Abbildung 4.18 veranschaulicht einige **Funktionsdefinitionen** der Programmiersprache C++, welche semantisch auf Basis der verwendeten Wissensrepräsentation abgebildet wurden. Die konkrete Modellierung von Funktionen im semantischen Netz wird anhand eines Beispiels in Abschnitt 5.2.4 ausgeführt.

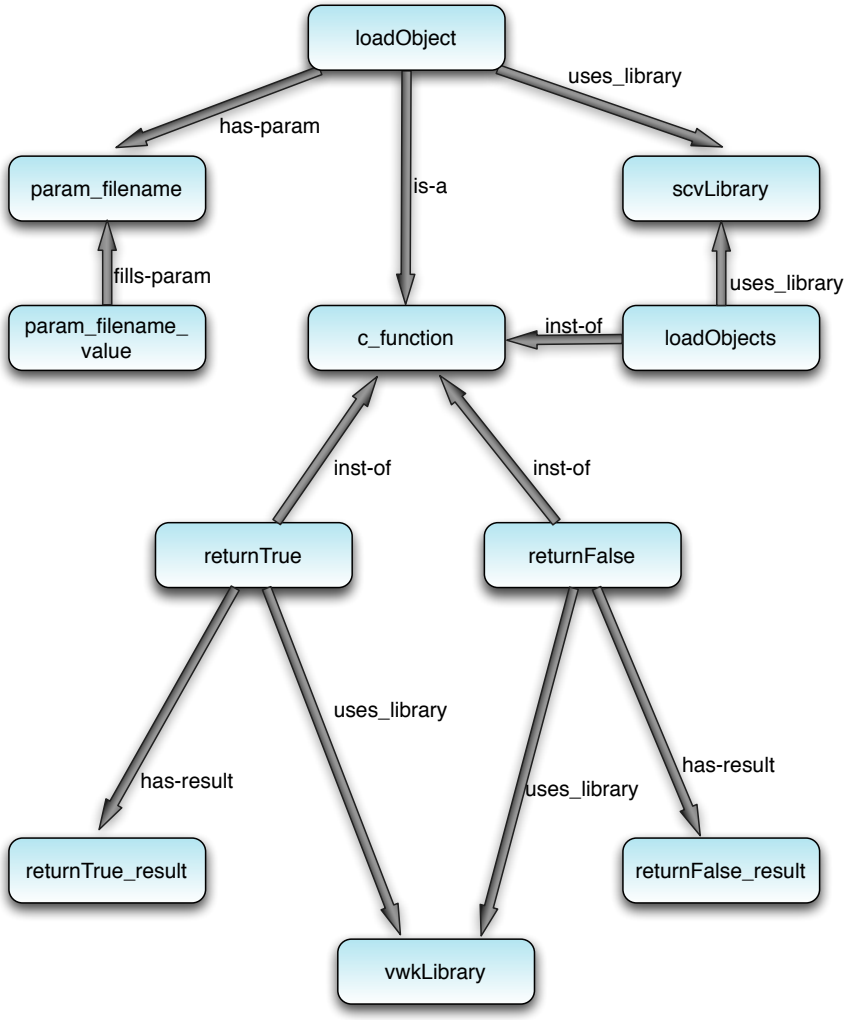


Abb. 4.18: Semantische Reflexion eines Funktionsaufrufes in der Programmiersprache C++ im semantischen Netzwerk.

Die speziellen Funktionen werden mittels Vererbung modelliert. Sie sind daher abgeleitet von einem Basiskonzept `c_function` und durch `is-a` Relation mit diesem verbunden. Es werden dabei sowohl parameterlose Funktionen (`returnTrue` und `returnFalse`), als auch Funktionen, welche einen Parameter benötigen (`loadObject`), abgebildet. Darüber hinaus besteht die Möglichkeit Funktionen zu modellieren, welche mehr als nur einen Parameter übergeben bekommen. Der korrekte Funktionsaufruf kann konstruiert werden, indem einem speziell dafür vorgesehenen **Slot** ein Index zugewiesen wird, sodass der Funktionsknoten weiß, in welcher Reihenfolge die Parameter sortiert werden müssen.

Die Funktionen werden zur Laufzeit von speziellen semantischen Traversern aufgerufen, welche das semantische Netzwerk auf der Suche nach spezifischen Knoten- und Relationstypen durchlaufen. Hat ein Traverser sein gewünschtes Ziel gefunden, konstruiert er den Funktionsaufruf, indem er ausgehend vom Funktionsknoten die Relationen für die benötigten Parameter sowie den Rückgabewert traversiert und im Falle der Parameter in die gewünschte Reihenfolge sortiert. Der fertig konstruierte Aufruf wird dann mit Hilfe der LIBFFI (FOREIGN FUNCTION INTERFACE) Bibliothek, welche Teil der GCC Distribution ist, aufgerufen. Die `uses-library`-Relation zeigt auf den Knoten, welcher das benötigte *shared object* enthält, das die konkrete Implementierung der Funktion beinhaltet.

4.3.3.5 Zeitliche Abfolgen

Die in Abschnitt 3.5 vorgestellten 13 Relationen können verwendet werden, um **zeitliche Abfolgen** semantisch zu modellieren. Abbildung 4.19 zeigt beispielhaft, wie zeitliche Prozesse in SCIVE modelliert werden. Zwei Sequenzen des Typs `temporal_sequence` werden instanziiert. Diese werden vom allgemeinen Typ `temporal_process` abgeleitet und initiieren die Berechnung der Simulationsmodule sowie die Synchronisation ihrer Ergebnisse. Nach der Initialisierung der `calculation_sequence` starten die verschiedenen Module ihre konkurrenten Berechnungsschleifen. Wenn sie diese beendet

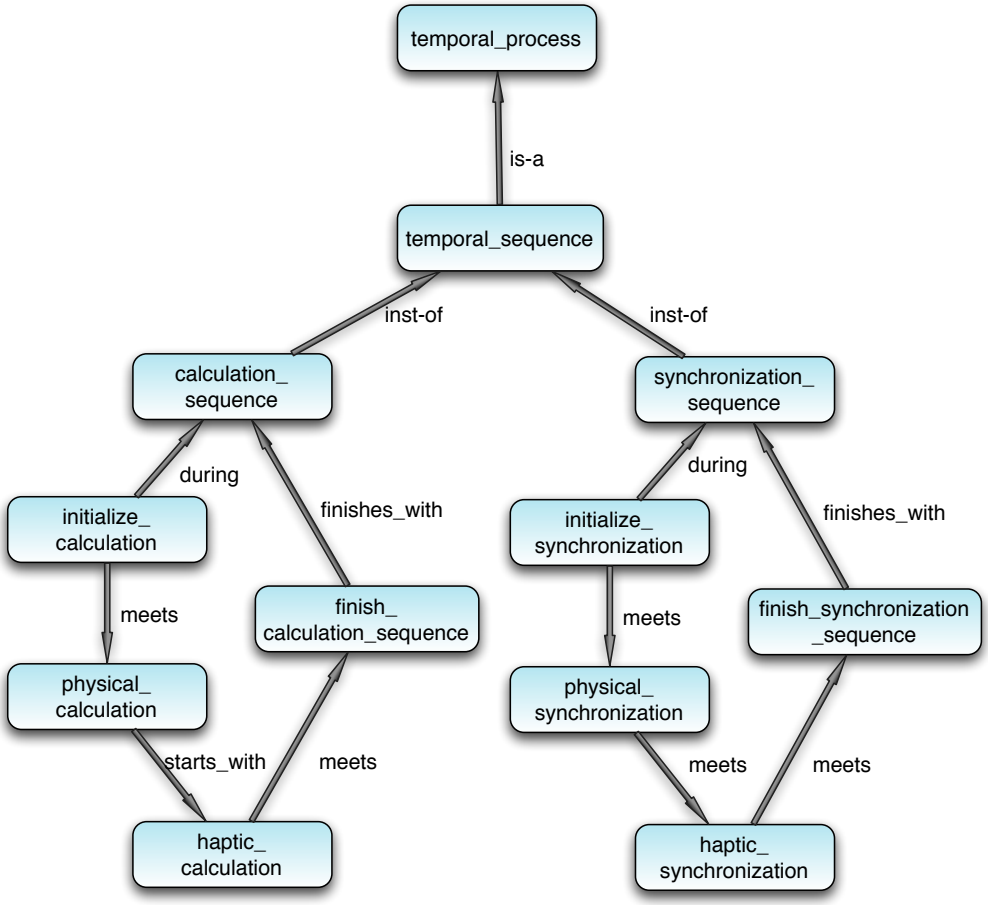


Abb. 4.19: Zeitliche Sequenzen von Aktionen modelliert in SCIVE mittels semantischer Reflexion

haben, wird die Funktion `finish_calculation_sequence` aufgerufen. Die zwei Sequenzen sind durch die `meets`-Relation verbunden, was bedeutet, dass die `synchroniazation_sequence` exakt mit dem Beenden der Berechnungen startet. In diesem Fall findet die Synchronisation der Daten also nicht konkurrent, sondern nacheinander statt. **Die Reihenfolge der Verarbeitung wird durch die Bedeutung der Relation bestimmt**.

Die Relationen zwischen den Knoten bestimmen die Art und Weise, auf die die Sequenz durch die Netzwerk-Traverser ausgeführt wird. Die Sequenzen können zur Laufzeit geändert werden, indem die **Netzwerk-Topologie**

durch eine Skriptschnittstelle verändert wird. Um die Reihenfolge innerhalb einer Sequenz zu ändern, müssen lediglich die Relationen ausgetauscht werden. Auf die abgebildeten Sequenzen bezogen, könnte beispielsweise die Relation `starts-with` zwischen den Knoten `physical_calculation` und `haptic_calculation` durch eine `meets`-Relation getauscht werden. Dies hätte dann in der Verarbeitung der Sequenz zur Folge, dass die physikalischen und die haptischen Berechnungen nicht mehr gleichzeitig starten, sondern nacheinander ausgeführt werden. Die Skriptschnittstelle wird durch das SWIG[2] Framework generiert, welches es auch ermöglicht, Schnittstellen für andere Skriptsprachen zu erzeugen (siehe [Beazley 2003] und [Beazley 1996]). Es besteht darüber hinaus die Möglichkeit, dass die aufgerufenen Funktionen selbst die Topologie verändern. In diesem Fall würde eine der aufgerufenen Berechnungen die Relationen zwischen den Knoten verändern. Details zu der Umsetzung der Skriptschnittstelle werden im Abschnitt 5.1 beschrieben. Die Modellierung von temporalen Sequenzen mittels einer konkreten Wissensrepräsentation wird weitergehend in Sektion 5.2.5 ausgeführt.

4.4 Zusammenfassung

Das zurückliegende Kapitel stellte die Integration von Konzepten und Prinzipien aus den Kapiteln 2 und 3 vor. Bevor die Hauptbereiche der modularen Simulationsarchitektur sowie der semantischen Modellierung beschrieben wurden, wurde das Framework SCIVE (Simulation Core for Intelligent Virtual Environments) als technische Basis der Integrationen vorgestellt. Für die beiden Bereiche wurden dann jeweils Anforderungen gestellt, ein Konzept entwickelt sowie die exemplarische Integration der Konzepte gezeigt. Für die modulare Simulationsarchitektur wurden die semantische Reflexion sowie das Actor Model in SCIVE integriert. Es wird die **Modularität erhöht**, da die Komponenten unabhängiger voneinander werden und über definierte Nachrichten miteinander kommunizieren. Alle Module arbeiten dabei auf derselben Datenstruktur – dem funktional erweiterbaren semantischen Netz.

[2] **SWIG**: **S**implified **W**rapper and **I**nterface **G**enerator

Im Bereich der semantischen Modellierung wurde gezeigt, wie diverse Bestandteile einer Simulation semantisch reflektiert werden können. Dazu zählen unter anderem die semantischen Traverser, Funktionen oder die mittels dem Allen-Kalkül modellierten temporalen Sequenzen. Durch die modulare Struktur des Actor Models, die auf die Simulationskomponenten übertragen wurde, sowie die umfangreiche semantische Modellierung, geht das entwickelte Konzept vor allem in Bezug auf die Modularität und Eigenständigkeit der Module, weiter als andere Systeme.

Das folgende Kapitel 5 zeigt, wie die in diesem Kapitel dargestellten konzeptuellen Umsetzungen konkretisiert und implementiert werden. Im Bereich der Architektur wird auf Erweiterungen des SCIVE-Frameworks eingegangen, während für die semantische Modellierung anhand detaillierter Listings dargestellt wird, wie die verschiedenen Inhalte konkret modelliert werden können.

KAPITEL

5

IMPLEMENTIERUNG

Die in Kapitel 4 vorgestellte Integration hat gezeigt, wie verschiedene Konzepte zu der modularen Simulationsarchitektur und zu der semantischen Modellierung virtueller Umgebungen beitragen können. Während diese auf abstrakter Ebene beschrieben wurden, werden im folgenden Kapitel exemplarisch die Implementierungen dargelegt.

Das Kapitel ist dabei in zwei Bereiche gegliedert. Beginnend mit den Arbeiten im Bereich der Modularität, werden Erweiterungen an der SCIVE-Architektur vorgestellt. Darauf folgend wird die semantische Modellierung virtueller Umgebungen und ihrer Logik im Bereich der temporalen Sequenzen, Funktionen und Systemkonfigurationen Schritt für Schritt in Listings ausgeführt. Diese zeigen, wie die verschiedenen Aspekte in der Beschreibungssprache SNIL auf deklarative Weise modelliert werden.

Teile der vorgestellten Arbeiten sind in [Fröhlich u. Latoschik 2008] und [Fröhlich u. a. 2009a] veröffentlicht.

5.1 Erweiterung der Architektur

In den folgenden Abschnitten wird zunächst der Ausbau der modularen Simulationsarchitektur vorgestellt. Ausgehend von dem in Abschnitt 4.1 eingeführten Framework SCIVE werden Erweiterungen dargelegt. Diese verfolgen das Ziel aus Abschnitt 1.2, der **Entwicklung einer modularen Simulationsarchitektur**. Dabei begegnet die Erweiterung um eine Skriptschnittstelle der geforderten Integration von bewährten Methoden der VR-Entwicklung.

5.1.1 Modularisierung der Skriptschnittstelle

Zur Erstellung des Skriptinterfaces wurde SWIG (siehe Abschnitt 4.3.3.5) verwendet. Hierbei handelt es sich um ein Programmierwerkzeug, welches in C oder C++ geschriebene Module für andere Programmiersprachen, insbesondere Skriptsprachen verfügbar macht. Als Zielsprache wird eine Reihe moderner und gängiger Skriptsprachen, wie zum Beispiel SCHEME oder PYTHON, unterstützt. SWIG unterstützt Zielsprachen, die in der Lage sind, C-Bibliotheken einzubinden. Die Mehrzahl moderner Programmiersprachen ist nur deswegen sinnvoll einsetzbar, da sie eben diese Möglichkeit besitzen. Fast alle betriebssystemnahen Bibliotheken sind in C oder einem Derivat geschrieben. Wären moderne Skriptsprachen nicht in der Lage auf diese Bibliotheken zuzugreifen, wären sie keinesfalls universell einsetzbar.

Das Einbinden solcher Module erfolgt in der Regel dadurch, dass der Quellcode der Quellbibliothek unverändert bleibt. Dabei werden zusätzliche Funktionen, sogenannte Wrapper, erstellt, die die ursprünglichen Funktionen kapseln. Diese dienen als Schnittstelle zwischen der Bibliothek und der Skriptsprache und können aus dieser heraus aufgerufen werden, Parameter übernehmen und Rückgabewerte liefern. Die **Wrapper-Funktionen** implementieren ihrerseits nur eine Schnittstellenanpassung und rufen die entsprechende Funktion in der Ausgangsbibliothek auf.

Das Erstellen der Schnittstelle wird von SWIG weitestgehend automatisiert – es muss lediglich die *Headerdatei* der Quellbibliothek um spezifische

Anweisungen erweitert werden. Alternativ hierzu kann die Datei auch einfach in einen speziellen von SWIG verwendeten Header eingebunden werden. SWIG erzeugt dann C-Quelltext, welcher die Wrapper für sämtliche Funktionen der Ausgangsbibliothek enthält. Darüber hinaus wird auch der benötigte Wrappercode für die ausgewählte Zieldatei erstellt. Die Erzeugung dieser Schnittstellen erfolgt ohne Zugriff auf den Quellcode der Ausgangsbibliothek. Die Wrapper werden mittels des plattformabhängigen *Compilers* übersetzt und entweder mit in die Quellbibliothek gelinkt oder in eine separate Bibliothek geschrieben. Die Nutzung in der Zielsprache bedingt keine weitere Laufzeitumgebung.

Die Grenzen von SWIG werden im Wesentlichen durch die **Mächtigkeit der Zielsprache** bestimmt. Viele Skriptsprachen unterstützen nicht alle Konzepte von C++. Ein Beispiel hierfür ist die Mehrfachvererbung, welche kaum in anderen Sprachen vorkommt. In solchen Fällen kann eine Anpassung des Quellcodes in der Ausgangsbibliothek oder eine zusätzliche Kapselung von Objekten und Funktionen von Nöten sein.

Der Vorteil dieser Methode, eine Schnittstelle zu erzeugen, liegt im Besonderen darin, dass verschiedene Sprachen angeboten werden können. Diese Sprachen können je nach Programmiererfahrung der Benutzer in verschiedenen Projekten angewendet werden. Eine prototypische Verwendung der Schnittstelle kombiniert mit der Programmiersprache Python wird im folgenden Abschnitt näher erläutert, wobei auch auf die Vorteile der verwendeten Sprache eingegangen sowie deren Auswahl begründet wird.

5.1.2 Erweiterung um eine Skript-Komponente

Wie bereits einleitend in Abschnitt 1.2 erwähnt, wurde eine Komponente entwickelt, die es erlaubt, über eine Skriptsprache Einfluss auf die laufende Simulation zu nehmen. Diese ermöglicht interaktive Zugriffe und Änderungen auf eine laufende Anwendung und erlaubt so ein Rapid Prototyping. Zu den Vorteilen einer solchen Vorgehensweise zählt unter anderem die Möglichkeit,

schnelle Korrekturen in der Software während der Laufzeit vorzunehmen, um kleine Fehler direkt korrigieren zu können. Dieses kann dem Entwickler sehr viel Zeit ersparen, da es nicht mehr notwendig ist, die Anwendung bei einem erkannten Fehler zu beenden, den Fehler zu beseitigen, die Anwendung zu kompilieren und neu zu starten.

Im Normalfall beziehen sich Änderungen durch Rapid Prototyping auf Sourcecode, der in einer Skriptsprache entwickelt wurde. In dem hier entwickelten System werden über eine Skriptschnittstelle Änderungen an der gemeinsamen Wissensbasis (dem semantischen Netz) vorgenommen. Bestimmte Zugriffsfunktionen auf das Netz werden für eine Skriptsprache exportiert, sodass es beispielsweise möglich wird, Knoten zur Laufzeit hinzufügen oder zu entfernen, also **Strukturänderungen an dem semantischen Netz** vorzunehmen. Es ist darüber hinaus auch möglich, Werte einzelner Knoten zu beeinflussen. Als Beispiel sei hier die Änderung der Gravitationskonstante der physikalischen Simulation genannt.

Prototypisch wurde eine Anbindung der Skriptsprache PYTHON vorgenommen. Bei PYTHON handelt es sich um eine interpretierte höhere Programmiersprache. Die Vorteile liegen darin, dass mehrere Paradigmen des Programmierens unterstützt werden. Es ist möglich *objektorientiert*, *aspektorientiert*[1] oder *funktional* zu programmieren. Darüber hinaus arbeitet PYTHON effizient und ist für Neueinsteiger einfach zu erlernen, da die Sprache über eine klare und übersichtliche Syntax verfügt. Dies war unter anderem eines der Ziele bei der Entwicklung der Sprache, daher kommt sie im Vergleich zu anderen Sprachen mit relativ wenig Schlüsselwörtern aus und die Syntax ist auf ihre Übersichtlichkeit hin optimiert.

In PYTHON wird darüber hinaus auch das bereits erwähnte Konzept „Alles ist ein Objekt“ aufgegriffen (vergleiche Abschnitt 2.3). Die Datentypen

[1] **Aspektorintierte Programmierung:** Bezeichnet ein spezielles Programmierparadigma der objektorientierten Programmierung. Der Ansatz konzentriert sich darauf, allgemeine Funktionen über multiple Klassen hinweg zur Verfügung zu stellen.

werden dynamisch vergeben. Es existiert also keine strenge Typisierung wie beispielsweise in C++. Der jeweilige Datentyp wird **an das Objekt gebunden** und nicht an die jeweils deklarierte Variable. Dieses Vorgehen ist unter anderem schon in älteren Sprachen wie LISP oder SMALLTALK zu finden.

In der vorliegenden Anwendung wurde durch den in Abschnitt 5.1.1 dargestellten Mechanismus eine Schnittstelle erzeugt, mittels welcher die simulierte Szene beeinflusst werden kann. Mit Hilfe dieser Schnittstelle können die an der Simulation beteiligten Komponenten angefragt werden. Die Komponenten stellen ein Objekt über das sogenannte *Singleton Pattern* bereit, welches eindeutig referenzierbar ist und gewisse Zugriffsfunktionen zur Verfügung stellt. Listing 5.1 zeigt ein kurzes Skript, in dem die oben genannten Operationen durchgeführt werden.

```
1  import scvInterface

3  l = libSCIVE_GetLibSCIVE()
4  g = sciveGraphicsComponentOSG_GetGraphicsComponentOSG()
5  p = scivePhysicsComponent_GetPhysicsComponent()
6  a = sciveFModSoundComponent_GetFModSoundComponent()

8  p.setGravity(0,-981,0)

10 a.play("elevator.wav")

12 l.parseConfiguration("simulationconfig.xml")
13 l.loadComponents()
14 l.loadData()
15 l.startFunctionTraverser("loadMultipleObjectsInSCIVE")
```

Listing 5.1: Zugriff auf verschiedene Module mittels PYTHON

Exemplarisch werden in Listing 5.1 zunächst verschiedene Komponenten (der Simulationskern, das Grafikmodul, die Physikkomponente und die Audioausgabe) der Reihe nach angefragt (Zeilen 3 bis 6). Die referenzierten Komponenten bieten diverse Funktionen an, welche auch zur Laufzeit aufgerufen werden können und verschiedene Effekte in der simulierten Szene hervorrufen. So wird im Beispiel die berechnete Gravitationskraft der Physikkomponente verändert sowie das einfache Abspielen einer Audiodatei aufgerufen (Zeilen 8 und 10). Zum Abschluss werden im Simulationskern selber einige Funktionen aufgerufen, welche eine Konfiguration, die dazugehörigen Komponenten sowie die zu simulierenden Daten laden (Zeilen 12 bis 15).

Neben den oben gezeigten Modulen steht auch das semantische Netz über die Skriptschnittstelle zur Verfügung. Diese Komponente stellt ebenfalls verschiedene Funktionen bereit, mit denen folgende beispielhafte Operationen möglich sind:

- Hinzufügen oder Entfernen von **Domänen**
- Hinzufügen oder Entfernen einzelner **Knoten**
- **Ändern von Werten** in Feldern
- **Topologische Änderungen** des Netzes

Es könnte also mittels der Skriptsprache eine komplette Szene instanziiert und simuliert werden. Listing 5.2 zeigt die Verwendung einiger der genannten Funktionen an einem Beispiel. Zunächst wird der Zugriff auf das semantische Netz, beziehungsweise die Komponente definiert (Zeile 3). In einem zweiten Schritt wird die Domäne `SemNet` angelegt (Zeile 5). Es werden im Anschluss zwei Knoten `Node1` und `Node2` eingeführt (Zeilen 7 und 8), die dann durch eine `is-a`-Relation miteinander verbunden werden (Zeile 10).

```
1  import scvInterface

3  s = sciveSemNetComponent_GetSemNetComponent()

5  d1 = s.CreateDomain("SemNet");

7  n1 = s.CreateNode("Node1", "SemNet");
8  n2 = s.CreateNode("Node2", "SemNet");

10 r1 = s.CreateRelation("n1", "n2", "is-a");
```

Listing 5.2: Zugriff auf die zentrale Datenbasis in PYTHON

In der Praxis eignet sich die Skriptschnittstelle eher dazu, in einem Rapid Prototyping Prozess **mit der virtuellen Umgebung zu interagieren**, als diese in seiner Gesamtheit in der Skriptsprache zu definieren. Die Definition der virtuellen Umgebung kann in einem separaten Prozess in einer anderen Umgebung erfolgen.

5.2 Semantische Modellierung

Während die vorherigen Abschnitte die durchgeführten Implementierungen im Bereich der modularen Simulationsarchitektur geschildert haben, setzen sich die folgenden Sektionen mit der semantischen Modellierung verschiedener Aspekte virtueller Umgebungen auseinander. Dabei werden die konzeptuellen Modellierungen aus Abschnitt 4.3.3 konkretisiert und die Implementierungen werden Schritt für Schritt in Listings ausgearbeitet. Beginnend mit der semantischen Modellierung virtueller Inhalte werden im weiteren Systemkonfigurationen, ikonische Formen, Funktionen und temporale Sequenzen besprochen.

5.2.1 Virtuelle Inhalte

Im Folgenden wird beschrieben, wie eine virtuelle Umgebung auf Basis der im Lauf der Arbeit beschriebenen Mechanismen modelliert werden kann. Listing 5.3 zeigt die grundlegende Definition eines semantischen Netzes. Der `semantic-net` Knoten (Zeile 1 und 6) umspannt alle Definition von Relationen und Knoten, die den Inhalt der virtuellen Umgebung erzeugen. Das abgebildete semantische Netz enthält bereits die Definition zweier Relationen, welche sehr häufig in semantischen Netzen verwendet werden. Die `is-a` Relation (Zeile 2) beschreibt eine klassische Vererbungshierarchie. Sie kann verwendet werden, um zwei Knoten zueinander in Beziehung zu setzen, und drückt aus, das Knoten 2 von Knoten 1 abgeleitet ist. Des Weiteren wird ausgedrückt, dass diese Relation **transitiv** ist. Wenn eine Reihe von Knoten (Typen) mittels dieser Relation verbunden werden, kann geschlussfolgert werden, dass der letzte Knoten in der Hierarchie eine Unterart aller anderen Knoten ist, welche über ihm in der Hierarchie angesiedelt sind. Die `inst-of` Relation (Zeile 5) beschreibt die Beziehung eines speziellen Individuums zu seinem generellen Typ. Werden zwei Knoten mittels `inst-of` miteinander verbunden, bedeutet dies, dass Knoten 2 vom Typ Knoten 1 ist.

```
1 <semantic-net>
2     <relationtype name="is-a" type="Default">
3         <transitive />
4     </relationtype>
5     <relationtype name="inst-of" type="inst-of"/>
6 </semantic-net>
```

Listing 5.3: Ausgangsbasis einer Szenendefinition auf Basis des semantischen Netzes

Um nun die erzeugte virtuelle Welt weiter mit Leben zu füllen, wird das vorliegende Netz zunächst in **Domänen** aufgeteilt. Listing 5.4 zeigt das vorherige Beispiel, welches nun um drei Domänen erweitert wurde. Die jeweiligen `subdomain`-Knoten (Zeilen 2 - 4) werden eingefügt und mittels eindeutiger Namen definiert, über welche sie auch im weiteren Gebrauch referenziert

werden können. Bei den drei beispielhaften Domänen handelt es sich um die zentrale Wissensbasis (`KR-Central`), die grafische Domäne (`GRAPHICS`) und die Domäne für die Definition physikalischer Knoten (`PHYSICS`). Die `KR-Central` Domäne beinhaltet im weiteren Verlauf die allgemeinen Definitionen von Objekten, welche in der virtuellen Umgebung verwendet werden.

```
<semantic-net>
    <subdomain name="KR-Central"/>
    <subdomain name="GRAPHICS"/>
    <subdomain name="PHYSICS"/>}
    <relationtype name="is-a" type="Default">
        <transitive />
    </relationtype>
     <relationtype name="inst-of" type="inst-of"/>
</semantic-net>
```

Listing 5.4: In Domänen aufgeteiltes semantisches Netz

Da nun die semantisch voneinander getrennten Domänen definiert sind, können auch Relationen beschrieben werden, die diese zueinander in Verbindung setzen. Wie in Listing 5.5 zu sehen ist, wurden dem semantischen Netz zwei weitere Relationen hinzugefügt.

```
<semantic-net>
    <subdomain name="KR-Central"/>
    <subdomain name="GRAPHICS"/>
    <subdomain name="PHYSICS"/>
    <relationtype name="is-a" type="Default">
        <transitive />
    </relationtype>
    <relationtype name="inst-of" type="inst-of"/>
    <relationtype name="has-graphics" type="Default"/>
    <relationtype name="has-physics" type="Default"/>
</semantic-net>
```

Listing 5.5: Semantisches Netz mit Relationen

Die Relationen `has-graphics` und `has-physics` (Zeilen 9 und 10) werden verwendet, um Entitäten der generellen `KR-Central` Domäne mit ihren spezialisierten modulabhängigen Realisierungen zu verbinden. Diese werden nun eingefügt, da das Grundgerüst für ihre Definition aufgebaut ist.

Listing 5.6 zeigt das aufgebaute Beispiel, welches um einen generellen Knoten mit zwei spezialisierten Ableitungen erweitert wurde. Die Knoten werden mittels des `node`-Eintrags (Zeilen 11, 14 und 19) eingefügt und ihnen wird jeweils ein Name zugewiesen. Innerhalb der `node` Umgebung werden die jeweiligen Subdomänen mittels des `in-subdomain` Tags (Zeilen 12, 15 und 20) definiert. Darüber hinaus können den Knoten noch `Slots` (`slot`) (Zeilen 16 und 21) hinzugefügt werden, welche spezielle Attribute enthalten, beispielsweise den Verweis auf eine Datei, die ein bestimmtes 3D-Modell für die grafische Visualisierung, oder eine Definition physikalischer Eigenschaften enthält.

```
1  <semantic-net>
2      <subdomain name="KR-Central"/>
3      <subdomain name="GRAPHICS"/>
4      <subdomain name="PHYSICS"/>
5      <relationtype name="is-a" type="Default">
6          <transitive />
7      </relationtype>
8      <relationtype name="inst-of" type="inst-of"/>
9      <relationtype name="has-graphics" type="Default"/>
10     <relationtype name="has-physics" type="Default"/>
11     <node name="landscape" type="Default">
12         <in-subdomain name="KR-Central"/>
13     </node>
14     <node name="landscape-gfx" type="Default">
15         <in-subdomain name="GRAPHICS"/>
16         <slot name="File" type="string"
17          inheritanceType="Attribut" value="room.iv"/>
18     </node>
```

```
19      <node name="landscape-phys" type="Default">
20          <in-subdomain name="PHYSICS"/>
21          <slot name="File" type="string"
22           inheritanceType="Attribut"
23           value="landscape-phys.xml"/>
24      </node>
25  </semantic-net>
```

Listing 5.6: Semantisches Netz mit spezialisierten Knoten

Da nun die Inhalte der virtuellen Umgebung mit allen spezialisierten Definition vorhanden sind, müssen diese zum Abschluss noch miteinander verbunden werden.

Wie in Listing 5.7 zu sehen ist, verbinden die durch die `relation` Tags eingefügten Relationen die generelle Definition des `Landscape`-Objektes mit seinen spezialisierten Repräsentationen in den Grafik- und Physik-Domänen. Eine Relation wird über den passenden Namen instanziiert und bekommt als Attribut eine `id` (Zeilen 25 bis 32). Innerhalb der Umgebung werden dann Start- und Endknoten der erzeugten Beziehung angegeben. Die betreffenden Knoten werden wiederum über ihre zugewiesenen eindeutigen Namen (nodeName) referenziert.

Das gezeigte Beispiel baut abschließend eine einfache virtuelle Welt auf, die durch die semantischen Beziehungen von generellen Entitäten mit deren speziellen Repräsentationen verbunden ist. Hierzu wurde ein bestimmter Satz Domänen, Relationen und Knoten definiert.

```
<semantic-net>
    <subdomain name="KR-Central"/>
    <subdomain name="GRAPHICS"/>
    <subdomain name="PHYSICS"/>
    <relationtype name="is-a" type="Default">
        <transitive />
    </relationtype>
    <relationtype name="inst-of" type="inst-of"/>
    <relationtype name="has-graphics" type="Default"/>
    <relationtype name="has-physics" type="Default"/>
    <node name="landscape" type="Default">
        <in-subdomain name="KR-Central"/>
    </node>
    <node name="landscape-gfx" type="Default">
        <in-subdomain name="GRAPHICS"/>
        <slot name="File" type="string"
         inheritanceType="Attribut" value="room.iv"/>
    </node>
    <node name="landscape-phys" type="Default">
        <in-subdomain name="PHYSICS"/>
        <slot name="File" type="string"
         inheritanceType="Attribut" value="phys.xml"/>
    </node>
    <relation typeName="has-gfx" id="1">
        <start-node nodeName="landscape"/>
        <end-node nodeName="landscape-gfx"/>
    </relation>
    <relation typeName="has-phys" id="2">
        <start-node nodeName="landscape"/>
        <end-node nodeName="landscape-phys"/>
    </relation>
</semantic-net>
```

Listing 5.7: Semantisches Netz mit Knoten und den dazugehörigen Relationen

5.2.2 Systemkonfigurationen

Ebenso wie die im vorherigen Abschnitt beschriebene semantische Modellierung von virtuellen Inhalten, kann auch eine Systemkonfiguration des eingesetzten Simulationssystems selbst auf semantischer Basis modelliert werden. Das Prinzip funktioniert analog, allerdings wird auch hierfür eine spezielle Domäne mit spezialisierten Relationen eingefügt. Listing 5.8 zeigt die spezielle Domäne `SC` (kurz für `System Configuration`) (Zeile 1) zusammen mit den benötigten Relationen `has-module` und `has-feature` (Zeilen 2 und 3). Die `has-module`-Relation wird verwendet, um ein bestimmtes Modul der Gesamtkonfiguration hinzuzufügen, während die `has-feature`-Relation bestimmt, ob ein Modul über eine spezielle Eigenschaft verfügt.

```
1 <subdomain name="SC"/>
2 <relationType name="has-module" type="Default"/>
3 <relationType name="has-feature" type="Default"/>
```

Listing 5.8: Domäne und Relationen zur Definition einer Systemkonfiguration

Die benötigten Knoten können zu der erstellten Domäne hinzugefügt werden. Listing 5.9 zeigt die Erstellung dreier Knoten. `System_Configuration` (Zeile 4) stellt den Wurzelknoten dieser Domäne dar. Mit ihm werden die an der Simulation beteiligten Module verbunden, welche ihrerseits Features hinzugefügt bekommen. Die Konfiguration beinhaltet im vorliegenden Beispiel ein Modul `Physics` (Zeile 7) für eine physikalische Simulation. Darüber hinaus wurde noch ein Knoten definiert, der eine bestimmte Eigenschaft darstellt. Dieser Knoten wurde mit dem Namen `Collision_Detection` (Zeile 9) erzeugt und ist vom Typ `feature`.

```
1  <subdomain name="SC"/>
2  <relationType name="has-module" type="Default"/>
3  <relationType name="has-feature" type="Default"/>
4  <node name="System_Configuration" type="Default" id="1">
5      <in-subdomain name="SC" />
6  </node>
7  <node name="Physics" type="module" id="2">
8      <in-subdomain name="SC"/>
9  <node "Collision_Detection" type="feature" id="3">
10     <in-subdomain="SC"/>
11 </node>
```

Listing 5.9: Erweiterung der Konfiguration um Knoten (Konfiguration, Modul und Feature)

In einem letzten Schritt werden, wie in Listing 5.10 gezeigt, die erstellten Knoten mit den definierten Relationen verbunden. `System_Configuration` wird mittels der `has-module`-Relation zum `Physics`-Knoten verbunden (Zeilen 14 bis 17), während dem `Physics`-Knoten das `Collision_Detection`-Feature mittels der `has-feature`-Relation zugeordnet wird (Zeilen 18 bis 21).

```
1  <subdomain name="SC"/>
2  <relationType name="has-module" type="Default"/>
3  <relationType name="has-feature" type="Default"/>

5  <node name="System_Configuration" type="Default" id="1">
6      <in-subdomain name="SC" />
7  </node>
8  <node name="Physics" type="module" id="2">
9      <in-subdomain name="SC"/>
10 </node>
11 <node "Collision_Detection" type="feature" id="3">
12     <in-subdomain="SC"/>
13 </node
```

```
14 <relation typeName="has-module" id="4">
15     <start-node nodeName="System_Configuration"/>
16     <end-node nodeName="Physics"/>
17 </relation>
18 <relation typeName="has-feature" id="5">
19     <start-node nodeName="Physics"/>
20     <end-node nodeName="Collision_Detection"/>
21 </relation>
```

Listing 5.10: Semantisches Netz zur Definition einer Systemkonfiguration

Auf die vorgestellte Art und Weise kann ein Simulationskern mit all seinen beteiligten Modulen semantisch modelliert werden. Die Definition liegt innerhalb der zentralen Wissensbasis vor und kann durch den Nutzer zur Laufzeit des Systems manipuliert werden. Dies ermöglicht es dem Anwender zum Beispiel, ein Modul aus der Simulation zu entfernen oder hinzuzufügen sowie einzelne Features der Module an- oder abzuschalten.

5.2.3 Ikonische Formen und Gesten

Die Semantischen Reflexion erlaubt zusätzlich die Modellierung von ikonischen Formen und formbeschreibenden Gesten. Die im Abschnitt 4.3.3 erwähnten Imagistic Description Trees lassen sich auf die vorliegenden Strukturen übertragen.

Abbildung 5.1 zeigt eine detaillierte Darstellung eines gebogenen Rohres übertragen auf das semantische Netz. Abgeleitet von einem Basisknoten werden zwei Segmente definiert, welche in diesem Fall in einem festen Winkel aneinander gesetzt werden. Basierend auf dieser abstrakten Formbeschreibung können auch formbeschreibende Gesten mittels des semantischen Netzes definiert und für eine Gestenerkennung abgeglichen werden.

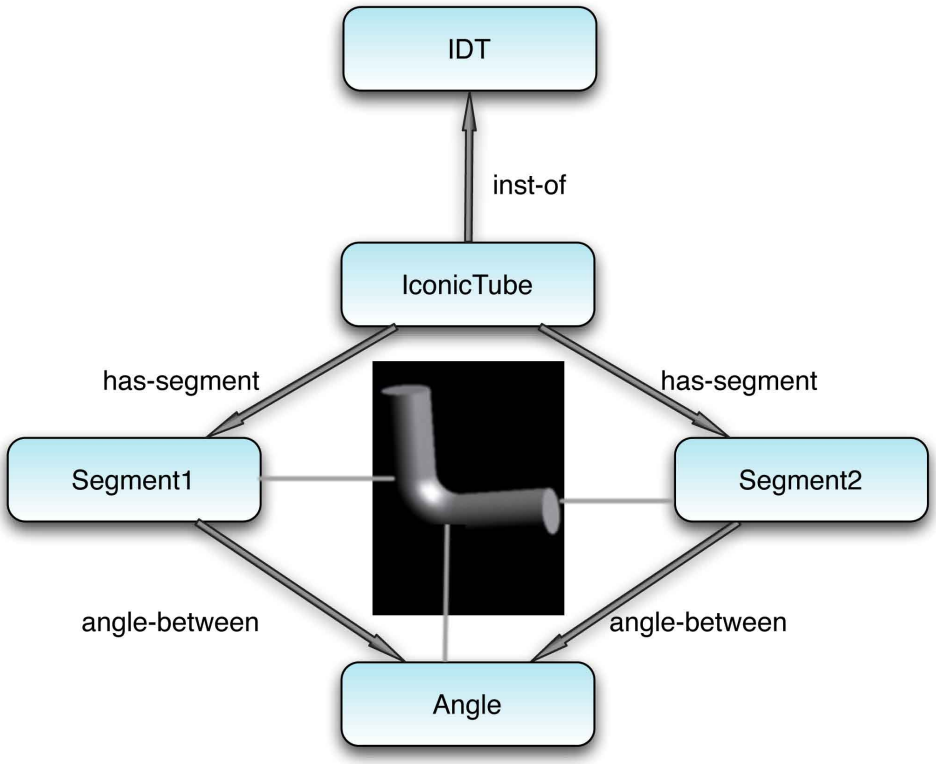


Abb. 5.1: Detaillierte Darstellung eines Rohres übertragen auf ein semantisches Netz

Übertragen auf diese Domäne kann die konkrete Implementierung einer ikonischen Geste abgebildet werden, wie in Listing 5.11 dargestellt ist. Im Zuge der Modellierung werden spezielle Knoten und Relationen nach dem in den vorangegangenen Abschnitten vorgestelltem Muster erstellt. Die ikonische Form kann so mit seinen Einzelteilen in hierarchischer Struktur abgebildet werden.

```
1 <subdomain name="Gesture Description"/>
2 <relationType name="has-segment" type="Default"/>
3 <relationType name="angle-between" type="Default"/>
4 <node name="IconicTube" type="Default" id="1">
5     <in-subdomain name="Gesture Description" />
6 </node>
```

```
<node name="Segment1" type="" id="2">
    <in-subdomain name="Gesture Description"/>
    <slot name="Length" type="double"
      inheritanceType="Attribute" value="22"/>
</node>
<node name="Segment2" type="" id="3">
    <in-subdomain name="Gesture Description"/>
    <slot name="Length" type="double"
      inheritanceType="Attribute" value="44"/>
</node>
<node name="Angle" type="" id="4">
    <in-subdomain name="Gesture Description"/>
    <slot name="Degrees" type="double"
      inheritanceType="Attribute" value="90"/>
</node>
<relation typeName="has-segment" id="5">
    <start-node nodeName="IconicTube"/>
    <end-node nodeName="Segment1"/>
</relation>
<relation typeName="has-segment" id="6">
    <start-node nodeName="IconicTube"/>
    <end-node nodeName="Segment2"/>
</relation>
<relation typeName="angle-between" id="7">
    <start-node nodeName="Segment1"/>
    <end-node nodeName="Angle"/>
</relation>
<relation typeName="angle-between" id="7">
    <start-node nodeName="Segment2"/>
    <end-node nodeName="Angle"/>
</relation>
```

Listing 5.11: Semantisches Netz zur Definition einer ikonischen Geste

5.2.4 Funktionen

Wie in Abschnitt 4.3.3 beschrieben, lassen sich auch Funktionsaufrufe auf einer semantischen Ebene modellieren. Abbildung 5.2 zeigt eine beispielhafte semantische Definition einer C++-Funktion, welche einen Parameter verwendet, mit dessen Hilfe Daten übergeben werden können – in diesem Fall ein Dateiname. Die Bibliothek, die die Funktion enthält, ist ebenfalls als Knoten modelliert und mittels einer Relation mit dieser verbunden.

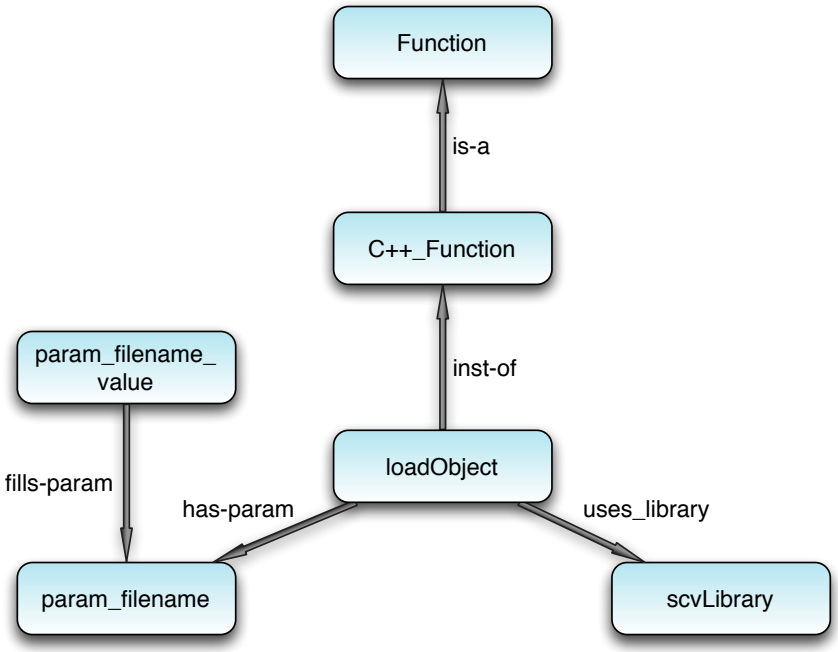


Abb. 5.2: Beispielhafte semantische Definition einer Funktion

Die Implementierung eines solchen Knotens wird im Folgenden dargestellt. Listing 5.12 zeigt zunächst die Definition der hierfür verwendeten grundlegenden Konzepte. Die Knoten `Function` und `C++_Function` werden definiert (Zeilen 2 bis 7) und mittels der bereits bekannten `is-a`-Relation miteinander verknüpft (Zeilen 8 bis 11). Die Domäne `Function Definitions` (Zeile 1), in der auch die beiden Knoten erstellt wurden, enthält fortan alle Knoten, die an der Definition von Funktionen beteiligt sind.

```
<subdomain name="Function Definitions"/>
<node name="Function" type="Default" id="1">
    <in-subdomain name="Function Definitions"/>
</node>
<node name="C++_Function" type="Default" id="2">
    <in-subdomain name="Function Definitions"/>
</node>
<relation typeName="is-a" id="3">
    <start-node nodeName="Function"/>
    <end-node nodeName="C++_Function"/>
</relation>
```

Listing 5.12: Semantische Definition allgemeiner Funktions-Knoten

Im Anschluss, dargestellt in Listing 5.13, wird die konkrete Funktion `LoadObject` instantiiert (Zeilen 1 bis 3) und als Instanz der `C++_Function` angelegt. Dies geschieht wie in den vorherigen Beispielen mittels Anlegen der `inst-of`-Relation (Zeilen 4 bis 7).

```
<node name="LoadObject" type="Default" id="4">
    <in-subdomain name="Function Definitions"/>
</node>
<relation typeName="inst-of" id="5">
    <start-node nodeName="LoadObject"/>
    <end-node nodeName="C++_Function"/>
</relation>
```

Listing 5.13: Semantische Definition einer konkreten C++ Funktion

Diese instantiierte Funktion wird anschließend mit dem notwendigen Parameter verknüpft, welcher seinerseits mit einem Wert gefüllt wird. Dieser Vorgang ist in Listing 5.14 dargestellt. Es werden zwei neue Relationen `has-param` und `fills-param` eingeführt (Zeilen 3 und 4), mit denen die erstellten Knoten `param_filename` und `param_filename_value` (Zeilen 13 bis 20) verknüpft werden.

```
1  <subdomain name="Function Definitions"/>

3  <relationtype name="has-param" type="Default"/>
4  <relationtype name="fills-param" type="Default"/>

6  <node name="param_filename" type="Default" id="6">
7      <in-subdomain name="Function Definitions" />
8  </node>
9  <node name="param_filename_value" type="Default" id="7">
10     <in-subdomain name="Function Definitions" />
11 </node>

13 <relation typeName="has-param" id="8">
14     <start-node nodeName="LoadObject" />
15     <end-node nodeName="param_filename" />
16 </relation>
17 <relation typeName="fills-param" id="9">
18     <start-node nodeName="param_filename_value" />
19     <end-node nodeName="param_filename" />
20 </relation>
```

Listing 5.14: Semantische Definition von Parametern einer C++ Funktion

Listing 5.15 zeigt die gesamte Definition des Funktionsaufrufes. Es wird ein weiterer Knoten `scvLibrary` eingeführt (Zeilen 23 bis 25), der einen Verweis auf die Programmbibliothek enthält, die die konkrete Implementierung der definierten Funktion enthält. Zu diesem Zweck wird die neue Relation `uses-library` definiert (Zeile 6), die die zwei Knoten verbindet (Zeilen 45 bis 48).

```
<subdomain name="Function Definitions"/>

<relationtype name="has-param" type="Default"/>
<relationtype name="fills-param" type="Default"/>

<relationtype name="uses-library" type="Default"/>

<node name="Function" type="Default" id="1">
    <in-subdomain name="Function Definitions" />
</node>
<node name="C++_Function" type="Default" id="2">
    <in-subdomain name="Function Definitions" />
</node>
<node name="LoadObject" type="Default" id="4">
    <in-subdomain name="Function Definitions" />
</node>
<node name="param_filename" type="Default" id="6">
    <in-subdomain name="Function Definitions" />
</node>
<node name="param_filename_value" type="Default" id="7">
    <in-subdomain name="Function Definitions" />
</node>
<node name="scvLibrary" type="Default" id="10">
    <in-subdomain name="Function Definitions" />
</node>

<relation typeName="is-a" id="3">
    <start-node nodeName="Function"/>
    <end-node nodeName="C++_Function"/>
</relation>
<relation typeName="inst-of" id="5">
    <start-node nodeName="LoadObject"/>
    <end-node nodeName="C++_Function"/>
</relation>
```

```
35 <relation typeName="has-param" id="8">
36     <start-node nodeName="LoadObject"/>
37     <end-node nodeName="param_filename"/>
38 </relation>

40 <relation typeName="fills-param" id="9">
41     <start-node nodeName="param_filename_value"/>
42     <end-node nodeName="param_filename"/>
43 </relation>

45 <relation typeName="uses-library" id="11">
46     <start-node nodeName="LoadObject"/>
47     <end-node nodeName="scvLibrary"/>
48 </relation>
```

Listing 5.15: Semantisches Netz zur Definition eines Funktionsaufrufes

Damit sind alle benötigten Informationen zur Definition einer C++ Funktion vorhanden. Die konkrete Funktion wird von einem abstrakten Konzept abgeleitet und die Parameter werden mit den entsprechenden Werten gefüllt. Wird der fertige Funktionsaufruf durch einen Traverser konstruiert, kann dieser mittels des in Abschnitt 4.3.3 eingeführten FOREIGN FUNCTION INTERFACE in der angegebenen Bibliothek inklusive seiner Parameter aufgerufen werden.

5.2.5 Temporale Sequenzen

Die in Abschnitt 3.5 vorgestellten temporalen Relationen sowie deren Anwendung (siehe Abschnitt 4.3.3.5), können auf ähnliche Art und Weise semantisch modelliert werden. Hierzu werden Sequenzen definiert, welche auf zeitlicher Ebene durch Relationen verknüpft werden, die die Intervallalgebra von Allen widerspiegeln. Die erwähnten Sequenzen enthalten ihrerseits wiederum Funktionsaufrufe, wie sie im vorherigen Abschnitt 5.2.4 vorgestellt wurden. Die Reihenfolge der Ausführung wird dabei von den verknüpfenden Relationen bestimmt.

Abbildung 5.3 zeigt die Definition einer einfachen Sequenz von Funktionen in definierter Reihenfolge. Bei dem vorliegenden Beispiel handelt es sich um eine Abfolge von Berechnungsfunktionen verschiedener Aspekte einer Simulation. An dieser Stelle werden lediglich physikalische und haptische Berechnungen durchgeführt. Es ist allerdings ohne weiteres möglich, weitere Funktionen, wie auditive und grafische Berechnungen, in diese Sequenz einzufügen. Die Funktionen werden in ihrer verknüpften Reihenfolge abgearbeitet. Je nach Relation geschieht dieses nacheinander oder parallel. Die Sequenz wird mittels der bekannten `is-a` und `inst-of`-Relationen von allgemeinen Konzepten abgeleitet.

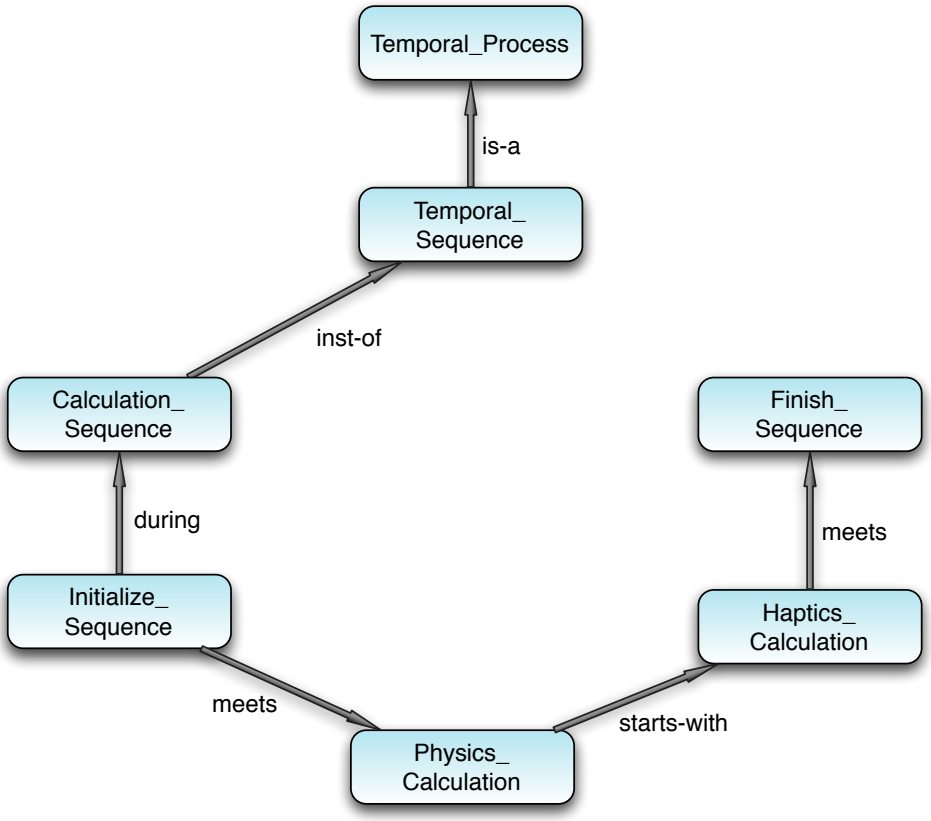


Abb. 5.3: Beispielhafte semantische Definition einer temporalen Sequenz. Die Relationen nach Allen bestimmen die Reihenfolge der Ausführung.

Im Folgenden wird gezeigt, wie die Knoten mit den entsprechenden Relationen definiert werden können. Die Definition der einzelnen Funktionen ist an dieser Stelle vereinfacht ohne ihre Parameter und Bibliotheksverweise, wie sie in Abbildung 5.2 gezeigt wurden, dargestellt. Sie werden hier als einfache Knoten innerhalb des semantischen Netzes gezeigt. Listing 5.16 zeigt, wie als erstes die Domäne `Temporal Processes` erstellt wird (Zeile 1). Anschließend werden die benötigten Knoten erstellt. Hierzu zählen die generellen Konzepte `Temporal Process` und `Temporal Sequence` (Zeilen 2 bis 7) sowie die konkreten Funktionen (Zeilen 8 bis 22).

```
1  <subdomain name="Temporal Processes"/>
2  <node name="Temporal Process" type="Default" id="1">
3      <in-subdomain name="Temporal Processes" />
4  </node>
5  <node name="Temporal_Sequence" type="Default" id="2">
6      <in-subdomain name="Temporal Processes" />
7  </node>
8  <node name="Calculation_Sequence" type="Default" id="3">
9      <in-subdomain name="Temporal Processes" />
10 </node>
11 <node name="Initialize_Sequence" type="Default" id="4">
12     <in-subdomain name="Function Definitions" />
13 </node>
14 <node name="Physics_Calculation" type="Default" id="5">
15     <in-subdomain name="Function Definitions" />
16 </node>
17 <node name="Haptics_Calculation" type="Default" id="6">
18     <in-subdomain name="Function Definitions" />
19 </node>
20 <node name="Finish_Sequence" type="Default" id="7">
21     <in-subdomain name="Function Definitions" />
22 </node>
```

Listing 5.16: Semantische Definition von Knoten einer temporalen Sequenz

Um nun die zeitliche Abfolge der Funktionsaufrufe definieren zu können, müssen die Relationen, mit ihrer festgelegten Semantik nach Allen, eingeführt werden. Listing 5.17 zeigt exemplarisch die Definition von drei Relationen entsprechenden den Allen'schen Beziehungen (Zeilen 2 bis 4).

```
1 <subdomain name="Temporal Processes"/>
2 <relationtype name="during" type="Default"/>
3 <relationtype name="starts-with" type="Default"/>
4 <relationtype name="meets" type="Default"/>
```

Listing 5.17: Semantische Definition von temporalen Relationen nach Allen

Mit diesen drei Relationen können die erstellten Knoten, wie in Listing 5.18 dargestellt, verbunden werden (Zeilen 2 bis 17). Da die Relationen nach Allen bidirektional zu betrachten sind, ist ihre Richtung von Bedeutung.

```
1  <subdomain name="Temporal Processes"/>
2  <relation typeName="during" id="8">
3      <start-node nodeName="Initialize_Calculation"/>
4      <end-node nodeName="Calculation_Sequence"/>
5  </relation>
6  <relation typeName="meets" id="9">
7      <start-node nodeName="Initialize_Calculation"/>
8      <end-node nodeName="Physics_Calculation"/>
9  </relation>
10 <relation typeName="starts-with" id="10">
11     <start-node nodeName="Physics_Calculation"/>
12     <end-node nodeName="Haptics_Calculation"/>
13 </relation>
14 <relation typeName="meets" id="9">
15     <start-node nodeName="Haptics_Calculation"/>
16     <end-node nodeName="Finish_Calculation"/>
17 </relation>
```

Listing 5.18: Semantische Verbindung von Knoten einer temporalen Sequenz

Die semantische Bedeutung, das heißt die Reihenfolge der Ausführung, wird durch die Verbindung mittels der Relationen nach Allen festgelegt. Die definierten Funktionen werden dann nach der Verarbeitung durch semantische Traverser ausgeführt.

5.3 Zusammenfassung

Dieses Kapitel hat – getrennt in die zwei Bereiche dieser Arbeit – gezeigt, wie die konzeptuell beschriebenen Integrationen aus Kapitel 4, im SCIVE-Framework implementiert wurden. Zunächst wurden Erweiterungen in SCIVE dargestellt, die das System um zusätzliche Funktionen ausbauen. Es wurde unter anderem die Erstellung einer modularen Skriptschnittstelle für die Programmiersprache Python vorgestellt. Die Schnittstelle ist jedoch so aufgebaut, dass auch andere Sprachen verwendet werden können. Im Bereich der semantischen Modellierung wurde ausgearbeitet, wie die verschiedenen Aspekte einer VR-Simulation mittels der deklarativen Sprache SNIL modelliert werden können. Dabei wurden die Modellierung virtueller Inhalte, semantischer Traverser, interaktiv generierter Objekte, Funktionen sowie zeitlicher Abfolgen gezeigt. Der technische Beitrag der vorgestellten Arbeiten besteht in der Erweiterung des SCIVE-Frameworks sowie der Übertragung und konkreten Modellierung der verschiedenen Simulationsaspekte in der Beschreibungssprache des semantischen Netzes. Durch die Modellierung der verschiedenen Bestandteile der Simulation in einem semantischen Netz wird ein Abstraktionsniveau erreicht, das über das anderer Architekturen hinaus geht.

Das folgende Kapitel 6 stellt anhand des Konstruktionsprojekts „Virtuelle Werkstatt“ den Transfer einiger der bislang vorgestellten Arbeiten dar. Dabei wird zunächst die technische Umgebung dargestellt, bevor auf die Umsetzungen im Bereich der wissensbasierten Computergrafik und der modularen Simulation der virtuellen Umgebung eingegangen wird.

KAPITEL

6

ANWENDUNGSBEISPIEL

Dieses Kapitel stellt den Transfer des entwickelten Frameworks in die Anwendungsdomäne der virtuellen Konstruktion vor. Dabei werden a) die Vorteile der verwendeten Methodik und b) deren Mächtigkeit demonstriert. Es werden verschiedene Aspekte im Kontext der Anwendungsdomäne hervorgehoben und deren Übertragung auf die vorgestellten Konzepte aus Kapitel 4 und Implementierungen aus Kapitel 5 gezeigt. Am Beispiel des Projekts **Virtuelle Werkstatt** werden konkrete Beispiele aus den Bereichen der Verarbeitung ikonischer Gesten und Formen, der semantischen Modellierung virtueller Umgebungen sowie der Modularisierung von VR-Simulationssystemen vorgestellt. Bevor näher auf diese Themen eingegangen wird, soll zunächst die technische Umgebung vorgestellt werden, die die Durchführung des Projekts in der AG WBS ermöglicht hat.

Teile der im folgenden Kapitel vorgestellten Arbeiten wurden von Fröhlich, Wachsmuth, u. Latoschik [2009b] und Biermann, Fröhlich, Latoschik, u. Wachsmuth [2007] veröffentlicht.

6.1 Technische Umgebung

Die technische Umgebung der AG WBS ist in einer dreiseitigen CAVE-artigen Mehrseitenprojektions-Installation verankert (siehe Abbildung 6.1). Auf jeder der Seiten werden zwei Bilder projiziert, eines für das rechte Auge und eines für das linke. Die beiden Bilder werden jeweils durch zirkuläre Polarisationsfilter vor den Projektoren wie auch an der zu tragenden Brille des Benutzers getrennt, um einen orientierungsunabhängigen Stereoeffekt zu erzeugen. Jedes der insgesamt sechs Bilder wird von einem einzelnen Rechner erzeugt, sodass allein für die Darstellung der Szene sieben Computer benötigt werden, wobei einer davon als Server der Anwendung fungiert und sowohl die Anwendungslogik verwaltet, wie auch die Verteilung der Szene auf die sogenannten Renderclients übernimmt. Die einzelnen Rechner sind untereinander über ein **Hochgeschwindigkeits-Netzwerk** (Infiniband) verbunden, welches einen sehr hohen Datendurchsatz (bis zu 20Gbit/s) bei einer sehr geringen Latenzzeit im niedrigen Millisekundenbereich erlaubt.

Für die Interaktion ist ein **Trackingsystem** der Firma A.R.T. im Einsatz. Diese Kameras senden infrarotes Licht aus, welches von kleinen Markern reflektiert wird, die der Benutzer in der CAVE am Körper trägt. In einer typischen Anwendung befinden sich dedizierte Markertargets an den Händen, um die Gestik des Benutzer erfassen zu können, sowie an einer Brille, damit die Perspektive immer korrekt für Anwender berechnet werden kann und stets einen realistischen immersiven Eindruck von der Szene vermittelt. Die Abbildung 6.1 zeigt die technische Umgebung der AG WBS und damit auch der Virtuellen Werkstatt im schematischen Überblick.

Um verschiedene Arten der Gestik verarbeiten zu können, ist ein einfaches optisches Tracking der Hände natürlich nicht ausreichend, da diese nur Trajektorien der Hände erfassen können. Will man aber auch beispielsweise Greif- oder Zeigegesten erkennen können, muss zusätzlich zu den Trajektorien auch noch die Fingerstellung des Benutzers erkannt werden. Dies wurde in der virtuellen Werkstatt durch **Datenhandschuhe** (Cybergloves) rea-

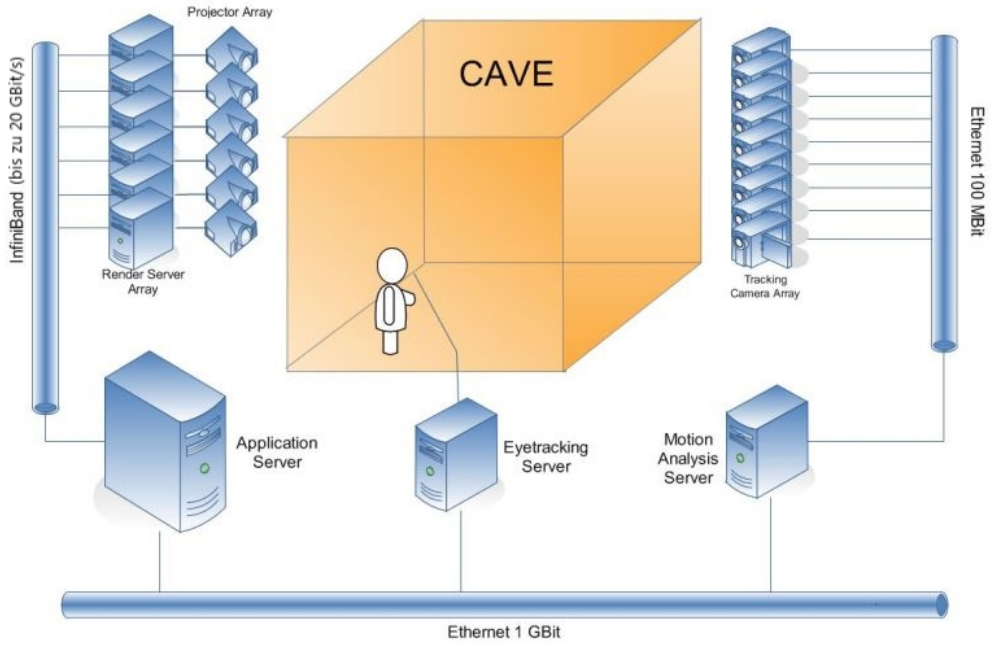


Abb. 6.1: Technische Umgebung der AG WBS im Überblick

lisiert, welche über den Widerstand der eingearbeiteten Bimetall-Streifen in der Lage sind, die Gelenkwinkel der Finger zu bestimmen. Gegen Ende des Projektes wurde auch ein optisches Trackingverfahren der Finger von der Firma A.R.T. erfolgreich exploriert, welches auch ein haptisches Feedback an den Fingerspitzen enthält.

Abbildung 6.2 zeigt eine Szene aus der CAVE der AG WBS, in der ein virtuelles Citymobil konstruiert wird. Die virtuelle Konstruktion wurde in verschiedenen Projekten erforscht und wird im folgenden Abschnitt am Beispiel der Virtuellen Werkstatt näher ausgeführt.

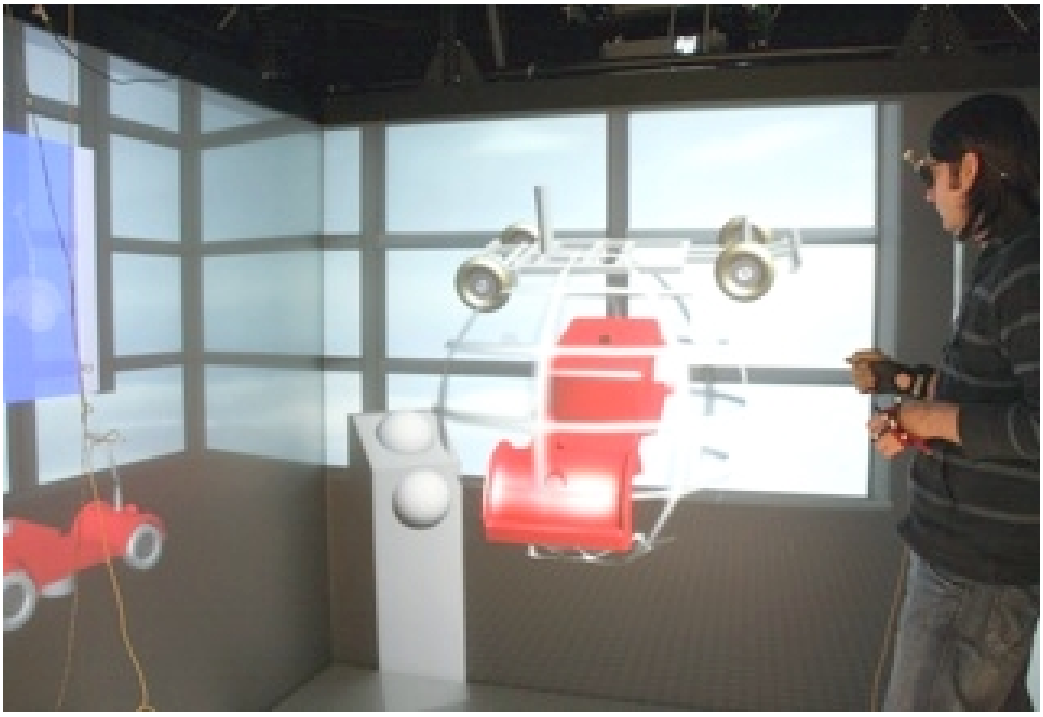


Abb. 6.2: Eine beispielhafte Szene in der CAVE der AG WBS beim Zusammenbau eines virtuellen Citymobils

6.2 Virtuelle Konstruktion

Die Domäne der virtuellen Konstruktion wird anhand des von der Deutschen Forschungsgemeinschaft geförderten Projekts **Virtuelle Werkstatt** [Jung u. a. 2002] ausgearbeitet. Dieses wurde im November 2001 mit dem Ziel begonnen, computergrafisch visualisierte dreidimensionale Modelle realer Konstruktionsteile in der Virtuellen Realität erprobbar zu machen. Das Projekt wurde in zwei Förderungsphasen unterteilt, von November 2001 bis Dezember 2004 sowie von Januar 2006 bis Dezember 2007. Während dieser zwei Förderungsphasen wurden viele technische Neuerungen auf dem Gebiet der Virtuellen Konstruktion erarbeitet. Zu diesen Neuerungen gehören unter anderem Arbeiten in den Bereichen der Künstlichen Intelligenz, des Virtuellen Prototypings, der multimodalen Mensch-Maschine Interaktion sowie der Softwareentwicklung für Simulationssysteme der Virtuellen Realität.

Die folgenden Abschnitte geben einen Überblick über die durchgeführten Arbeiten, wobei unter anderem Neuerungen im Bereich der wissensbasierten Computergrafik vorgestellt werden. Folgend werden dann die durchgeführten Arbeiten in Bezug auf Software-Engineering für intelligente virtuelle Umgebungen vorgestellt und eine Umsetzung einiger Aspekte des Projektes auf die hier erarbeite Software-Architektur gezeigt.

6.2.1 Wissensbasierte Computergrafik

Die virtuelle Werkstatt verfolgt den Ansatz der wissensbasierten Computergrafik, wobei die virtuellen Bauteile mittels den in Abschnitt 3.1 eingeführten **semantic entities** mit zusätzlichem Wissen angereichert werden. Diese semantischen Informationen enthalten sowohl Wissen über mögliche intelligente Verbindungsstellen der Teile, als auch über Parameter, die verwendet werden, um Teile innerhalb der Anwendung, also in Echtzeit, zu verändern. So sind zum Beispiel Skalierungen der Teile bei Erhalt ihrer Verbindungsstellen möglich. Ein weiterer Aspekt, der die Echtzeitfähigkeit der virtuellen Werkstatt fördert, ist die multimodale Mensch-Maschine-Interaktion. Diese ermöglicht durch eine Integration von gesprochener Sprache und verschiedenen Arten der Gestik eine natürliche Interaktion mit dem System. Diese Aspekte werden in den folgenden Abschnitten näher beleuchtet.

Abb. 6.3: Zweihändige Skalierung eines Bauteiles in der Virtuellen Werkstatt

Die intelligenten Bauteile verfügen außerdem über Wissen darüber, ob sie sich in ihrer Form verändern können, und wenn ja, in welcher Form und Dimension. So sind in der virtuellen Werkstatt **parametrische Skalierungen**

möglich. Diese bewirken, dass zum Beispiel ein Reifen, bei Erhalt der Dimension und relativen Position seiner Verbindungsstelle am Bauteil, skaliert werden kann. Bei diesem konkreten Beispiel werden also nur die „Felge“ und der „Gummireifen“ in ihrer Größe verändert (siehe Abbildung 6.3). Der Verbindungsport bleibt in seiner Ausdehnung unverändert. Diese Skalierungen werden durch einen hierarchischen Aufbau der Teile ermöglicht. Die **Bauteilhierarchie** wird in dem deklarativen XML-Format VPML (Variant Part Markup Language [Biermann u. Jung 2004] spezifiziert. Erweiterungen des VPML-Formates über *Templates* [Biermann u. a. 2007] erlauben eine Simulation krümmbarer Extrusionsformen in Echtzeit, durch Formauflösung im Krümmungsbereich. Die Kombination variabler Segmentanzahlen und sukzessiver Elementtransformationen ermöglicht Krümmungen unter Erhalt des korrekten optischen Eindrucks. Abbildung 6.4 zeigt Bauteile, die mittels Template-Definitionen mit verschiedenen Ausgangsformen und Krümmungsparametern erzeugt wurden.

Abb. 6.4: Unterschiedliche gekrümmte Formen und Krümmungswinkel [Biermann u. a. 2007]

Die in Abbildung 6.4 dargestellten Bauteile können durch die in Abschnitt 5.2.3 gezeigten Mechanismen erzeugt werden. Durch die Abbildung ikonischer Gesten kombiniert mit natürlicher Sprache können die Formen der zu erzeugenden oder zu verändernden Rohre interpretiert werden. Mit der Aussage „Gib mir **so** ein Rohr“ in Zusammenhang mit der entsprechenden Geste wird ein korrespondierendes Rohr erzeugt, während die Aussage „Biege das Rohr“ ein bestehendes Rohr der Geste entsprechend verändert. Die zu Grunde liegenden formbeschreibenden IDTs werden auf semantischer Ebene abgespeichert und den Bauteilen im Szenengraphen hinzugefügt. Abbildung 6.5 zeigt die Abbildung eines beispielhaften Rohres auf semantischer Ebene. Die Formbeschreibung wird auf semantischer Ebene gespeichert und kann auch für Vergleiche auf **ikonischer Basis** verwendet werden. Mit der Aussage „Nimm dieses Rohr“ [+ ikonische Geste] wird dasjenige virtuelle Rohr selektiert, das in der abgespeicherten Formbeschreibung der durchgeführten Geste am ehesten gleicht. Nur die Geste allein wäre nicht eindeutig genug, da sie auch zum Generieren von Bauteilen verwendet wird. In Kombination mit der sprachlichen Äußerung ergibt sich eine für das System eindeutige Instruktion. Der Unterschied zwischen den Kommandos zum Generieren und Selektieren von Bauteilen liegt also in der sprachlichen Komponente.

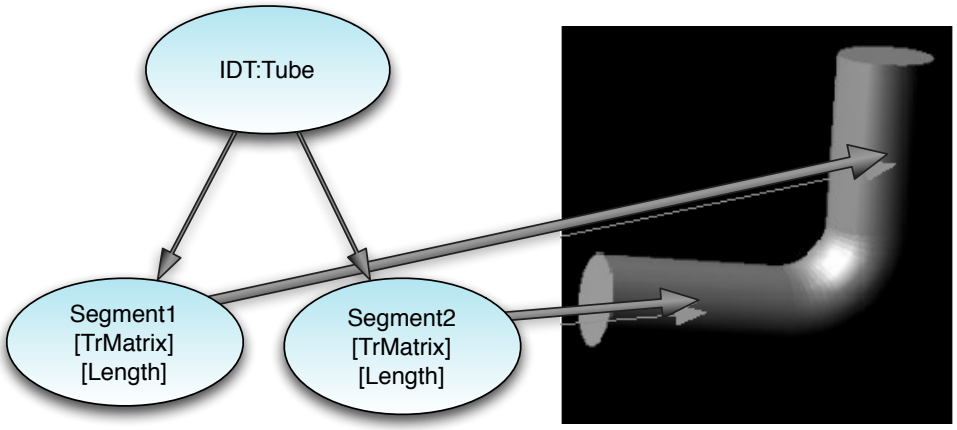


Abb. 6.5: Abbildung der Formbeschreibung eines gebogenen Rohres auf semantischer Ebene [Fröhlich u. a. 2009a]

6.2.2 Modularisierung und semantische Modellierung

Das in dieser Arbeit erweiterte modulare Simulationsframework SCIVE bietet die Möglichkeit, verschiedene Simulationsmodule, die unabhängig voneinander – innerhalb ihrer eigenen Simulationsschleifen – laufen, miteinander interagieren zu lassen und ihre Simulationsergebnisse untereinander auszutauschen sowie eventuell auftretende Konflikte zwischen diesen Modulen aufzulösen (vgl. Kapitel 4).

In der Virtuellen Werkstatt kommen eine Grafikkomponente und ein physikalisches Simulationssystem zum Einsatz, wodurch es möglich ist, eine **physikalisch animierte Umgebung** darzustellen. Die wissensbasierte Modellierung wird wie in Kapitel 5 beschrieben, durch ein semantisches Modul in Form des funktional erweiterbaren semantischen Netzes realisiert. Auf diesem Netz werden diverse Aspekte einer laufenden Simulation nach dem Prinzip der semantischen Reflexion abgebildet. Dabei werden basierend auf SNIL sowohl Simulationsinhalte wie die oben beschriebenen Bauteile, als auch die Anwendungslogik in Form von zeitlichen Abfolgen modelliert. Ähnliche Ansätze sind unter anderem bei [Lugrin u. Cavazza 2007] und [Kalogerakis u. a. 2006] zu finden. Die Abbildung des semantischen Wissens folgt dabei der in Abschnitt 4.2 beschriebenen Integration der semantischen Reflexion.

Die Integration der Simulationskomponenten auf der Ebene des Actor Models funktioniert wie in Abschnitt 4.2 dargestellt. Die einzelnen an der Simulation beteiligten Module, wie etwa die physikalische Simulation oder die Grafik, werden dabei an eine **Aktor-Schicht** angeschlossen, welche die auf dem Modell basierende Kommunikation mit den anderen Modulen realisiert. Die Module berechnen ihre Daten in einer eigenen Simulationsschleife und tauschen sich im Anschluss daran über definierte Nachrichten aus. Die Berechnung geschieht angepasst auf die jeweiligen Bedürfnisse der Komponenten. Während der Grafiksimulation 25 Bilder pro Sekunde ausreichen, um flüssig dargestellt zu werden, muss die Berechnungsfrequenz in dem physikalischen Modul sehr viel höher sein, um realistische Ergebnisse zu liefern.

Die **Anwendungslogik** wird entsprechend der im Abschnitt 4.3 konzeptuell ausgearbeiteten Integration vorgenommen. Dabei werden die verschiedenen Aspekte der semantischen Modellierung temporaler und funktionaler Aspekte entsprechend der in Sektion 5.2 exemplarisch dargestellten Implementierung durchgeführt. Die Konstruktion von Funktionsaufrufen auf semantischer Ebene sowie die Funktionsweise von semantischen Traversern wurden in den Abschnitten 5.2.4 beziehungsweise 4.3.3 gezeigt.

Die an das SCIVE-Framework angeschlossenen Komponenten umfassen die Grafikengine OpenSG, die Physikengine ODE sowie eine Anbindung der FMod Audio-Bibliothek und das Wissensrepräsentationsmodul in Form des semantischen Netzes. Darüber hinaus wurde eine modulare Skriptschnittstelle für verschiedene Sprachen entwickelt, deren Umsetzung in Abschnitt 5.1 detailliert beschrieben wurde. Wie dort dargelegt, wurde die Implementierung für die Programmiersprache Python vorgenommen, wobei durch den Aufbau der Schnittstelle auch andere Sprachen zu realisieren sind. Die Verarbeitung von Interaktionen durch den Benutzer werden ebenso in ein eigenes Modul gekapselt.

Zur Simulation der virtuellen Werkstatt und anderer virtueller Umgebungen steht also eine **Menge in sich abgeschlossener Aktoren** zur Verfügung, die ihre Daten in die gemeinsame Wissensrepräsentation integrieren, in der auch die Logik der Anwendung abgebildet wird und die über die Skriptschnittstelle manipuliert werden kann. Die Menge an Aktoren ist in folgender Liste zusammengefasst, während die zusammengefasste Reflexion der Daten mit der angeschlossenen Skriptschnittstelle in Abbildung 6.6 illustriert ist.

Aktoren, die zu der Simulation beitragen:

1. **Grafik** – OpenSG
2. **Physik** – ODE
3. **Audio** – FMod
4. **Wissensrepräsentation** – semantisches Netz
5. **Interaktion** – Verarbeitung von Eingaben

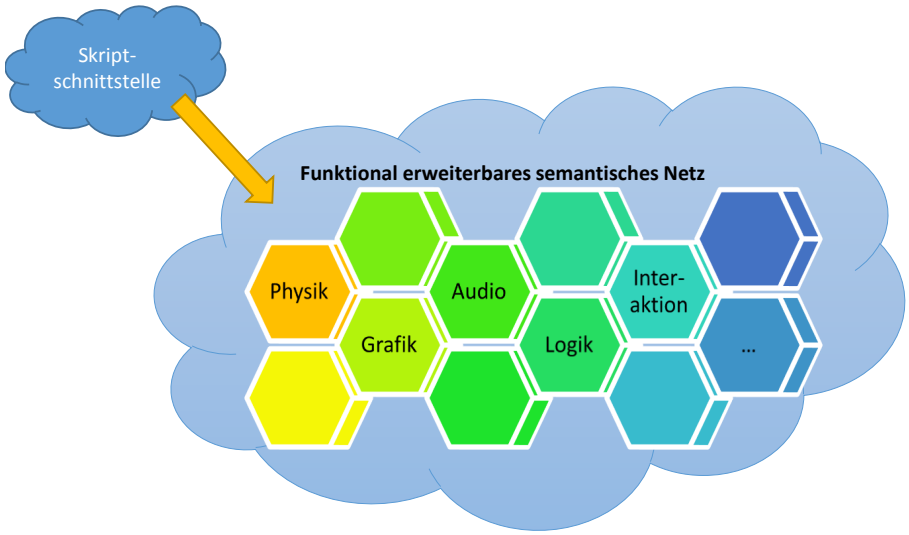


Abb. 6.6: Eine Zusammenfassung der Datenhaltung mit der Interaktion durch Skripte, die die gemeinsame Wissensrepräsentation manipulieren. Es werden die Daten der Aktoren sowie die Logik der Anwendung abgebildet.

6.3 Zusammenfassung

Dieses Kapitel hat die Anwendung einiger Aspekte dargestellt, die im Laufe der Arbeit vorgestellt wurden. Nach der Einführung in die technische Umgebung geschah dies am Beispiel der virtuellen Konstruktion in dem Projekt Virtuelle Werkstatt.

In der Virtuellen Werkstatt können komplette Konstruktionen frei gebaut und exploriert werden. Die multimodale Interaktion erlaubt integrierte Eingaben mittels Sprache und Gestik, für den Greif- und Fernraum (für Teile in direkter Reichweite des Benutzers und weiter entfernte), insbesondere auch mit **ikonischen Gesten**. Dabei sind sowohl die Repräsentationen der Bauteile, ihrer Verbindungen und parametrischen Veränderungen, als auch weitestgehend die Anwendungslogiken über deklarative XML-Formate beschrieben und in einer **gemeinsamen Wissensrepräsentation abgebildet**. Die parametrischen Veränderungen umfassen heterogene Skalierungen und Transformationen der Bestandteile. Die virtuelle Konstruktion erlaubt den vollständigen Aufbau komplexer zusammengesetzter Baugruppen, wobei alle Anpassungen über die Parameter im kompletten Aufbau weiterhin möglich sind.

Durch die Arbeiten im Bereich der Softwareentwicklung für VR-Systeme wurde die Virtuelle Werkstatt **modularisiert** und um weitere Komponenten erweitert, welche in Form der physikalischen Simulation sowie der Interaktion über verschiedene Skriptsprachen, neue Funktionalitäten mit sich bringen. Dabei kamen Konzepte aus Kapitel 4 und Implementierungen, die in Kapitel 5 vorgestellt wurden zum Einsatz. Durch den modularen Aufbau und der Art der Modellierung konnte eine **flexible Umsetzung** der Inhalte vorgenommen werden, die sich der diversen Funktionalitäten der einzelnen Komponenten bedient.

KAPITEL

7

RESÜMEE

„It's fine to celebrate success but it is more important to heed the lessons of failure."

Bill Gates

Bezugnehmend auf das oben stehende Zitat ist die retrospektive Betrachtung von Fehlern und Anfälligkeiten von mindestens ebenso zentraler Bedeutung wie die Umsetzung derer Aspekte, welche sich als erfolgreich und vielversprechend heraus gestellt haben. Im abschließenden Kapitel werden sowohl die Vorteile und erfolgreichen Aspekte, als auch die eventuellen Schwachstellen zusammengefasst. Darüber hinaus werden noch weitergehende Denkanstöße und Ausblicke auf mögliche folgende Arbeiten gegeben.

Die Zielsetzung der vorliegenden Arbeit in Erinnerung rufend, wurden in Abschnitt 1.2 die folgenden zwei Punkte genannt, um eine längere Persistenz von VR-Anwendungen zu ermöglichen:

1. **Entwicklung einer modularen Simulationsarchitektur:**
   (a) Verwendung von austauschbaren Komponenten, die ihre Daten in einen konsistenten Weltzustand integrieren
   (b) Integration von bewährten Methoden aus dem Bereich der VR-Entwicklung, um den Anwender mit vertrauten Funktionen auszustatten
2. **Semantische Modellierung virtueller Umgebungen**
   (a) Abstraktion der Simulationsinhalte und Anwendungslogik auf einer semantischen Ebene, um die Implementierung unabhängig von den verwendeten Modulen zu gestalten.

Im Folgenden wird an Hand dieser Ziele resümiert, welche Aspekte im Zuge der Arbeit durchgeführt wurden. Zu den jeweiligen Punkten wird eine Bewertung abgegeben, in der analysiert wird, inwieweit sich diese bewährt haben, oder an welcher Stelle Potential für Verbesserungen gegeben ist. Im Anschluss wird die Arbeit durch eine abschließende Betrachtung beendet.

7.1 Eine modulare Simulationsarchitektur

Wie in Abschnitt 1.2 dargelegt wurde, sollte es durch die hohe Modularität der Systemarchitektur ermöglicht werden, einzelne Simulationskomponenten austauschen zu können, ohne die gesamte Simulation neu konzipieren zu müssen. Die umgesetzten Schritte in diesem Bereich werden im Folgenden zusammengefasst, wobei diese in zwei Abschnitte unterteilt werden. Zunächst wird auf die Umsetzung der modularen Architektur eingegangen, um im Anschluss die Integration bewährter Methoden der VR-Entwicklung zu beschreiben, welche den Endanwender mit ihm bekannten Mechanismen ausstatten.

7.1.1 Architektur

Für die Umsetzung der modularen Systemarchitektur wurde das Framework SCIVE (Simulation Core for Intelligent Virtual Environments) erweitert. Es

tragen diverse Aspekte zur Wirkungsweise des Systems bei. Ein wichtiger Bestandteil der Architektur ist das Prinzip der semantischen Reflexion. Dieses wurde theoretisch in Abschnitt 3.2 eingeführt sowie in seiner exemplarischen Integration in das Framework in Sektion 4.2.3 näher erläutert. Die Reflexion sieht eine komplette Beschreibung aller Bestandteile der Simulation in einer einheitlichen semantischen Repräsentation vor. Dabei werden sowohl Inhalte wie auch Teile der Anwendungslogik reflektiert. Durch die in Domänen aufgeteilten Beschreibungen wird eine **Abstraktion der Implementierung und der Datenbeschreibung** erreicht. Resultierend hieraus ist im Falle eines Austausches einer Komponente lediglich ein kleiner Teil der Simulation neu zu gestalten.

Einen weiteren wichtigen Aspekt der Modularisierung stellt die Integration des Actor Models in SCIVE dar. Die Theorie des Modells wurde in Abschnitt 2.3 dargestellt, während die konzeptuelle Integration in das entwickelte System in Abschnitt 4.2.3 näher erläutert wurde. Wie in diesen Kapiteln beschrieben, ist das Modell in sich modular aufgebaut. Durch dessen inhärente Struktur und Übertragung des Konzeptes auf die Simulationsmodule in SCIVE wird ein hoher Grad an Modularität erreicht. Wie in der Integration beschrieben, werden die verschiedenen Komponenten in eine **Aktor-Schicht** gekapselt, welche die grundlegenden Funktionen eines Aktors gemäß dem Modell abbildet. Die Schnittstelle für ein Modul umfasst dabei lediglich vier Funktionen, die implementiert werden müssen.

7.1.1.1 Bewertung

Zusammenfassend lässt sich in Bezug auf die Zielsetzung sagen, dass es gelungen ist, eine Systemarchitektur zu entwerfen, die einen hohen Grad an Modularität aufweist. Durch die Abstraktion der Daten- und Modulbeschreibung auf einer gemeinsamen semantischen Schicht, die nach dem Prinzip der semantischen Reflexion die gesamte Simulation abbildet und unabhängig von konkreten Implementierungen der einzelnen Module agiert, ist eine Austauschbarkeit einzelner Komponenten gegeben.

Die Integration des Actor Models auf der Ebene der Simulationsmodule stellt einen vielversprechenden Ansatz dar. Die Übertragung des Modells auf die Struktur des semantischen Netzes bietet einige Vorteile gegenüber der Implementierung in früheren oft proprietären Programmiersprachen. Unter anderem wird die Portabilität des Simulationssystems erhöht und damit die Lebensdauer der entwickelten virtuellen Umgebungen verlängert. Darüber hinaus ist die Schnittstelle der Aktor-Schicht relativ einfach gehalten. Diese ermöglicht es, mit relativ geringem Aufwand, neue Module an die Architektur anzuschließen und bringt somit eine hohe **Flexibilität** mit sich.

7.1.2 Integration von VR-Methoden

Die Integration bewährter VR-Methoden in das erstellte Framework stellt einen weiteren wichtigen Aspekt dar. Es wurden gängige Funktionen eingebunden, welche Benutzer mit einer hohen Wahrscheinlichkeit aus anderen Frameworks kennen. Hierdurch werden eine gewisse **Vertrautheit und Konsistenz** geschaffen.

Es wurde eine Skript-Schnittstelle geschaffen, mit deren Hilfe ein Rapid Prototyping ermöglicht wird. Die Schnittstelle ermöglicht folgende Interaktionsmöglichkeiten:

1. Hinzufügen oder Entfernen von Daten in der Wissensrepräsentation
2. Ändern von Werten in der Wissensrepräsentation
3. Topologische Änderungen des semantischen Netzes
4. Zugriff auf die Aktoren und ausgewählte Funktionen
5. An- und Abschalten von Aktoren

Die Implementierung der Schnittstelle ist so gestaltet, dass verschiedene Skriptsprachen angebunden werden können. Ihre Umsetzung wurde in Abschnitt 5.1 erläutert. Die Verarbeitung von Skripten ist in der erstellten

modularen Architektur als eigenständiges Modul entwickelt und somit nicht eng an das Grafikframework gekoppelt worden. Die Manipulation der Daten erfolgt ausschließlich in der gemeinsamen Wissensbasis, welche die gesamte Anwendung reflektiert.

Die Struktur der verwendeten Wissensbasis ist an die eines **Szenengraphen** angelehnt, wie er in den meisten VR-Frameworks verwendet wird, sodass einige bewährte Methoden zum Einsatz kommen. Die gewählte Struktur des semantischen Netzes ist jedoch mächtiger als die eines Szenengraphen, da sie unter anderem nicht auf eine azyklische Struktur beschränkt ist. Es lassen sich Operationen eines Szenengraphen auf die Netzstruktur transferieren. So ist der Mechanismus der Traversierung ein übertragenes Konzept. Die Integration der semantischen Traverser ist in Abschnitt 4.3.3 beschrieben.

7.1.2.1 Bewertung

Die Integration der beschriebenen Methoden und Konzepte hat sich als erfolgreich herausgestellt. Die Umsetzung eines **Rapid Prototyping** Mechanismus gibt dem Anwender ein Werkzeug, mit dessen Hilfe er schnell verschiedene Sachverhalte in einer Simulation umsetzen und ändern kann. Die Möglichkeit interaktiv in die Simulation einzugreifen, um beispielsweise Module je nach Bedarf an- und abzuschalten, schafft ein erhöhtes Maß an Flexibilität in der Entwicklung von virtuellen Umgebungen. Hierdurch können Ressourcen geschont werden oder bestimmte Effekte in der Welt erzeugt werden.

Die Übertragung von Traverseroperationen von der Struktur des Szenengraphen auf das verwendete semantische Netz ist ebenso erfolgversprechend. Die Mächtigkeit der Traverser geht allerdings weit über die üblicherweise verwendeten Operationen auf einem Szenengraphen hinaus, was wiederum eine erhöhte Vielfalt der denkbar möglichen Operationen schafft.

7.2 Abstraktion der Simulationsinhalte und Anwendungslogik

Die Abstraktion der Simulationsinhalte wie auch der Anwendungslogik stellte ein wichtiges Ziel dieser Arbeit dar. Die Verbindung dieses Bereiches mit der Software-Systemarchitektur für VR-Systeme ist ein noch wenig erforschtes Gebiet. Daher sollte an dieser Stelle auch exploriert werden, ob sich bestimmte Strukturen und Modelle eignen und wenn ja, inwieweit sich diese in eine modulare Architektur einbinden lassen.

Zu den verwendeten Methoden aus dem Bereich der **Künstlichen Intelligenz**, zählen semantische Netzwerke und die Traverseroperationen, welche auf diesen durchgeführt werden können. Die Modellierung verschiedener Inhalte auf einer semantischen Netzstruktur wird konzeptuell in Abschnitt 4.3 ausgearbeitet. Die konkrete Umsetzung diverser Aspekte der Simulationsinhalte und der Anwendungslogik wird in Sektion 5.2 dargelegt. Durch den Einsatz der semantischen Traverser wird es dabei auch ermöglicht, Funktionsaufrufe in der semantischen Struktur zu definieren und zur Laufzeit der Anwendung zu konstruieren.

Ein weiteres integriertes Konzept ist **Allens Intervallalgebra**. Während die theoretischen Grundlagen in Sektion 3.5 erläutert wurden, wurde dessen konzeptuelle Anwendung in Abschnitt 4.3.3 ausgeführt. Allens 13 temporale Relationen kommen in der vorliegenden Arbeit bei der Definition der Anwendungslogik zum Einsatz. Hierbei werden verschiedene Ereignisse in einer definierten zeitlichen Abfolge miteinander verknüpft, wodurch an den Stellen, an denen dies notwendig oder von Vorteil ist, eine temporale Logik von Ereignissen geschaffen werden kann.

7.2.1 Bewertung

Die Integration der beschriebenen Konzepte und Methoden hat sich in ihren jeweiligen Bereichen als überwiegend positiv herausgestellt.

Die Verwendung des **semantischen Netzes** bietet einen mächtigen Ansatz, mit dessen Hilfe sehr flexible Anwendungen gestaltet werden können. Auf der konzeptuellen Ebene ist dies eine geeignete Struktur für diese Art von Architektur. Im einzelnen Anwendungsfall kann die Modellierung allerdings umständlich sein. Dadurch, dass alle Aspekte der Simulation auf dem semantischen Netz reflektiert werden, kann es für den Anwendungsentwickler unter Umständen mühsam sein, diese Vielzahl an Daten zu modellieren. An dieser Stelle wäre noch ein geeignetes Entwicklungswerkzeug von Nöten. Hierfür würde sich ein grafischer Editor anbieten, der dem Entwickler einige Arbeit abnimmt, bzw. ihm eine schnellere und übersichtlichere Arbeitsweise ermöglicht. Solche Editoren finden beispielsweise im Bereich der Computerspiele-Entwicklung bereits eine breite Anwendung.

Die Umsetzung von Allens Intervallalgebra stellt einen guten Mechanismus zur Regelung von zeitlichen Verhältnissen dar. Die Verarbeitung der **temporalen Abläufe** geschieht über die bereits beschriebenen Traverser, welche das semantische Netz gezielt nach den entsprechenden Relationen absuchen und dementsprechende Operationen in der genannten Reihenfolge abarbeiten. An den geeigneten Stellen bieten Allens Relationen einen hohen Grad an Flexibilität, ohne dabei zu restriktiv oder zu unkonkret zu werden. Die strikte Regelung von zeitlichen Abläufen kommt in einer Simulation dann zum Tragen, wenn es zeitkritische Faktoren gibt, bei denen es auf die Reihenfolge der Ausführung ankommt. An Stellen, wo dies nicht von Bedeutung ist, müssen die Relationen nicht eingesetzt werden.

7.3 Eine abschließende Betrachtung

In den zurückliegenden Kapiteln wurde die Erstellung einer modularen Simulationsarchitektur und die darauf basierende semantische Modellierung virtueller Umgebungen vorgestellt. Durch die Übertragung der Aktor-Struktur auf die beteiligten Simulationsmodule wird ein **Grad an Modularität** erreicht, der über andere Systeme hinaus geht. Durch die Schaffung definierter Schnittstellen wird es vereinfacht, Module auszutauschen. Die Integration der Skriptschnittstelle schafft außerdem die Möglichkeit, zur Laufzeit in die Anwendung einzugreifen. Damit erreicht die Architektur den Stand anderer verwandter Systeme und stattet den Anwender mit ihm bekannten Methoden der VR-Entwicklung aus. Die Schnittstelle geht insofern über den Stand der Forschung hinaus, als das sie modular gestaltet ist und die Verwendung verschiedener Programmiersprachen erlaubt.

Die **Abstraktion der Anwendung** wird durch die semantische Modellierung nach dem Prinzip der semantischen Reflexion durchgeführt. Dieser Methode folgend werden alle Aspekte der Simulation auf semantischer Ebene modelliert und sind leicht zugänglich – sowohl für die beteiligten Komponenten, wie auch für den Anwendungsentwickler. Diese Art der Modellierung geht über die in der VR-Entwicklung gängige Methode der semantischen Anreicherung im Szenengraphen hinaus und ermöglicht eine umfassende Erstellung virtueller Umgebungen. Die Modellierung der Anwendungslogik auf semantischer Ebene ist ein neu entwickeltes Konzept, das in dieser Art und in dem vorliegenden Umfang weiter geht als andere Systeme.

Kritisch zu betrachten bleibt die Erstellung der semantischen Modellierung, die auf Grund ihres Umfangs sehr zeitaufwändig ist. Ohne ein **fortschrittliches Entwicklungswerkzeug** stellt dies eine hohe Hürde für die alltägliche Entwicklung von umfassenden virtuellen Umgebungen dar. Ein solches Werkzeug wäre ein wichtiger Baustein zukünftiger Entwicklungen. Außerdem bleibt offen, inwieweit die Kommunikation unter den Modulen durch den Austausch von Nachrichten und die Integration der Daten in

die einheitliche Wissensrepräsentation die Performanz in großen virtuellen Umgebungen beeinflusst und die Konsistenz der Daten gewährleistet. Eine diesbezügliche Evaluation bleibt zukünftigen Arbeiten vorbehalten. So bleibt zusammenfassend zu sagen, dass durch die vorgenommenen Arbeiten ein System entstanden ist, das konzeptuell gut aufgestellt ist, aber auf im Hinblick auf seine Gebrauchstauglichkeit Raum für Verbesserungen und Evaluationen lässt.

Konstatierend sticht das entwickelte System durch seine Struktur und den damit verbundenen Umfang der möglichen Modellierung hervor. Durch die modulare Struktur sowie die unabhängige semantische Modellierung wird es ermöglicht, veraltete Komponenten durch neuere zu ersetzen, um so eine längere Persistenz von Anwendungen sicherzustellen. Obwohl dies auf Grund der oben genannten offenen Punkte vielleicht nicht der Weisheit letzter Schluss ist, bleibt doch zu hoffen, dass die vorgestellten innovativen Arbeiten weitere Forschungen auf diesem Gebiet inspirieren – getreu dem Motto „Inspiration durch Innovation“.

LITERATURVERZEICHNIS

[Agha 1986]
AGHA, Gul: *Actors: a model of concurrent computation in distributed systems*. Cambridge, MA, USA : MIT Press, 1986

[Allard u. a. 2004]
ALLARD, Jeremie ; GOURANTON, Valerie ; LECOINTRE, Loick ; LIMET, Sebastien ; MELIN, Emmanuel ; RAFFIN, Bruno ; ROBERT, Sophie: FlowVR: a Middleware for Large Scale Virtual Reality Applications. In: *Proceedings of the Euro-Par 2004 Conference*, 2004, S. 496 bis 505

[Allen 1983]
ALLEN, James F.: Maintaining Knowledge about Temporal Intervals. In: *Communications of the ACM* 26 (1983), November, Nr. 11, S. 832–843

[Beazley 1996]
BEAZLEY, David M.: SWIG: An easy to use tool for integrating scripting languages with C and C++. In: *Proceedings of the 4th USENIX Tcl/Tk workshop*, 1996, S. 129–139

[Beazley 2003]
BEAZLEY, David M.: Automated scientific software scripting with SWIG. In: *Future Generation Computer Systems* 19 (2003), Nr. 5, S. 599–609

[Biermann u. a. 2007]
BIERMANN, Peter ; FRÖHLICH, Christian ; LATOSCHIK, Marc E. ; WACHSMUTH, Ipke: Semantic Information and Local Constraints for Parametric

Parts in Interactive Virtual Construction. In: *Proceedings of the 8th International Symposium on Smart Graphics*. Kyoto, 2007, S. 124–134

[Biermann u. Jung 2004]
BIERMANN, Peter ; JUNG, Berhard: Variant Design in Immersive Virtual Reality: A Markup Language for Scalable CSG Parts. Articulated Motion and Deformable Object. In: *Proceedings of the AMDO-2004, Palma de Mallorca, Spain, Springer (LNCS 3179)*, 2004, S. 123–133

[Blach u. a. 1998]
BLACH, Roland ; LANDAUER, Jürgen ; RÖSCH, Angela ; SIMON, Andreas: A highly flexible virtual reality system. In: *Future Generation Computer Systems* 14 (1998), Nr. 3–4, 167–178. `citeseer.ist.psu.edu/396677.html`

[Brockhaus 1998]
BROCKHAUS: Brockhaus: Die Enzyklopädie (verschiedene Beiträge). In: F.A. BROCKHAUS AG, Bibliographisches I. (Hrsg.): *Brockhaus: Die Enzyklopädie*. 20. überarb. u. aktualisierte Aufl. Bibliographisches Institut der F.A. Brockhaus AG, 1998. – Erscheinungszeitraum 1997-1999

[Broderick 1982]
BRODERICK, D.: *The Judas Mandala*. Pocket Books, 1982 (Pocket science fiction)

[Bues u. a. 2008]
BUES, M ; GLEUE, T ; BLACH, R: Lightning-dataflow in motion. In: *Proceedings of the IEEE VR 2008 workshop SEARIS-Software Engineering and Architectures for Interactive Systems*, 2008

[Burns u. Osfield 2004]
BURNS, Don ; OSFIELD, Robert: Tutorial: Open scene graph A: introduction tutorial: Open scene graph B: examples and applications. In: *Virtual Reality, 2004. Proceedings.* IEEE, 2004, S. 265–265

[Cavazza 2000]
Cavazza, Marc: High-Level Interpretation in Dynamic Virtual Environments. In: *Applied Artificial Intelligence* 14 (2000), Nr. 1, S. 125–144

[Charniak 1985]
Charniak, Eugene: *Introduction to Artificial Intelligence*. Pearson Education, 1985

[Church 1951]
Church, A.: *Annals of Mathematical Studies*. Bd. 6: *The Calculi of Lambda-Conversion*. Princeton : Princeton University Press, 1951. – (second printing, first appeared 1941)

[Cruz-Neira u. a. 1992]
Cruz-Neira, Carolina ; Sandin, Daniel J. ; DeFanti, Thomas A. ; Kenyon, Robert V. ; Hart, John C.: The CAVE: audio visual experience automatic virtual environment. In: *Communications of the ACM* 35 (1992), Nr. 6, S. 64–72

[Dudenredaktion 2006]
Dudenredaktion, Dudenredaktion: *Duden - Die deutsche Rechtschreibung. Bd. 1.* 24. völlig neu bearb. u. erw. Mannheim : Dudenverlag, Bibliographisches Institut & F.A. Brockhaus, 2006

[Fellner u. a. 2009]
Fellner, Dieter W. ; Behr, Johannes ; Bockholt, Ulrich: Instantreality – A Framework for Industrial Augmented and Virtual Reality Applications. In: *Proceedings of the 2nd Sino-German Workshop on Virtual Reality & Augmented Reality in Industry*, 2009, S. 78–83

[Fischbach u. a. 2011]
Fischbach, Martin ; Wiebusch, Dennis ; Giebler-Schubert, Anke ; Latoschik, Marc E. ; Rehfeld, Stephan ; Tramberend, Henrik: SiXton's curse – Simulator X demonstration. In: *Virtual Reality Conference (VR)* IEEE, 2011, S. 255–256

[Fröhlich u. a. 2009a]

FRÖHLICH, Christian ; BIERMANN, Peter ; LATOSCHIK, Marc E. ; WACHSMUTH, Ipke: Processing Iconic Gestures in a Multimodal Virtual Construction Environment. In: M. DIAS, M. M. W. (Hrsg.) ; BASTOS, R. (Hrsg.): *Gesture-Based Human-Computer Interaction and Simulation*. Berlin : Springer, 2009, S. 187–192

[Fröhlich u. Latoschik 2008]

FRÖHLICH, Christian ; LATOSCHIK, Marc E.: Incorporating the Actor Model into SCIVE on an Abstract Semantic Level. In: *Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), Proceedings of the IEEE Virtual Reality 2008 Workshop*, 2008, S. 61–64

[Fröhlich u. a. 2009b]

FRÖHLICH, Christian ; WACHSMUTH, Ipke ; LATOSCHIK, Marc E.: Die Virtuelle Werkstatt - Multimodale Interaktion für intelligente Virtuelle Konstruktion. In: *In J. Gausemeier, M. Grafe (Hrsg.), 8. Paderborner Workshop Augmented und Virtual Reality in der Produktentstehung*. Paderborn HNI, 2009, S. 241–255

[Fröhlich u. Wachsmuth 2013]

FRÖHLICH, Julia ; WACHSMUTH, Ipke: The Visual, the Auditory and the Haptic – A User Study on Combining Modalities in Virtual Worlds. In: *Virtual Augmented and Mixed Reality. Designing and Developing Augmented and Virtual Environments*. Springer, 2013, S. 159–168

[Goldberg u. Robson 1983]

GOLDBERG, Adele ; ROBSON, David: *Smalltalk-80: the language and its implementation*. Boston, MA, USA : Addison-Wesley Longman Publishing Co., Inc., 1983

[Heilig 1962]

HEILIG, Morton L.: *Sensorama simulator, U.S.patent no.3050870*. August 1962

[Heilig 2014]
HEILIG, Morton L.: *Webseite von Morton Heilig.* `http://www.mortonheilig.com/InventorVR.html`, 2014. – Zuletzt besucht am 22.04.2014

[Heumer u. a. 2005]
HEUMER, G. ; SCHILLING, M. ; LATOSCHIK, M. E.: Automatic data exchange and synchronization for knowledge-based intelligent virtual environments. In: *Proceedings of the IEEE VR2005.* Bonn, Germany, March 2005, S. 43–50

[Hewitt u. a. 1973]
HEWITT, Carl ; BISHOP, Peter ; STEIGER, Richard: A Universal Modular ACTOR Formalism for Artificial Intelligence. In: *IJCAI*, 1973, S. 235–245

[Jung 1994]
JUNG, Bernhard: Exploiting the Actor Model for Reasoning in the Building Blocks World (Extended Abstract). In: *In J. Kunze & H. Stoyan (Hrsg.): KI-94 Workshops*, 1994, S. 28f

[Jung u. a. 2002]
JUNG, Bernhard ; LATOSCHIK, Marc E. ; BIERMANN, Peter ; WACHSMUTH, Ipke: Virtuelle Werkstatt. In: *1. Paderborner Workshop Augmented Reality / Virtual Reality in der Produktentstehung.* Paderborn : HNI, 2002, S. 185–196

[Jung u. Wachsmuth 1995]
JUNG, Bernhard ; WACHSMUTH, Ipke: Situierte Instruktionsverarbeitung im CODY Virtuellen Konstrukteur. In: *KI-95 Activities: Workshops, Posters, Demos*, Gesellschaft für Informatik e.V., 1995, S. 332–334

[Kalogerakis u. a. 2006]
KALOGERAKIS, Evangelos ; MOUMOUTZIS, Nektarious ; CHRISTODOULAKIS, Stavros: Coupling ontologies with graphics content for Knowledge Driven Visualization. In: *Proceedings of the IEEE Virtual Reality Conference*, 2006

[Kopp u. a. 2003]
KOPP, S. ; JUNG, B. ; LESSMANN, N. ; WACHSMUTH, I.: Max – A Multimodal Assistant in Virtual Reality Construction. In: *KI-Künstliche Intelligenz* 4 (2003), S. 11–17

[Kranstedt u. a. 2006]
KRANSTEDT, Alfred ; LÜCKING, Andy ; PFEIFFER, Thies ; RIESER, Hannes ; STAUDACHER, Marc: Measuring and Reconstructing Pointing in Visual Contexts. In: SCHLANGEN, David (Hrsg.) ; FERNANDEZ, Raquel (Hrsg.): *Proceedings of the brandial 2006 - The 10th Workshop on the Semantics and Pragmatics of Dialogue*. Potsdam : Universitätsverlag Potsdam, 2006, S. 82–89

[Kuck u. a. 2008]
KUCK, Roland ; WIND, Juergen ; RIEGE, Kai ; BOGEN, Manfred: Improving the AVANGO VR/AR Framework - Lessons Learned. In: *5. GI AR/VR Workshop, Magdeburg*, 2008

[Lanier 1992]
LANIER, Jaron: Virtual Reality: The Promise of the Future. In: *Interactive Learning International* 8 (1992), Nr. 4, S. 275–279

[Latoschik u. a. 1999]
LATOSCHIK, M. E. ; JUNG, B. ; WACHSMUTH, I.: Multimodale Interaktion mit einem System zur Virtuellen Konstruktion. In: BEIERSDÖRFER, K. (Hrsg.) ; ENGELS, G. (Hrsg.) ; SCHÄFER, W. (Hrsg.): *Informatik '99, - Informatik überwindet Grenzen, 29. Jahrestagung der Gesellschaft für Informatik, Paderborn, 5.10 - 9.10.1999*. Berlin : Springer, 1999, S. 88–97

[Latoschik 2002]
LATOSCHIK, Marc E.: Designing Transition Networks for Multimodal VR-Interactions Using a Markup Language. In: *Proceedings of the Fourth IEEE International Conference on Multimodal Interfaces ICMI'02, Pittsburgh, Pennsylvania*, IEEE, 2002, S. 411–416

[Latoschik u. a. 2005]
LATOSCHIK, Marc E. ; BIERMANN, Peter ; WACHSMUTH, Ipke: Knowledge in the Loop: Semantics Representation for Multimodal Simulative Environments. In: *Proceedings of the 5th International Symposium on Smart Graphics*, 2005, S. 25–39

[Latoschik u. Fröhlich 2007a]
LATOSCHIK, Marc E. ; FRÖHLICH, Christian: Semantic Reflection for Intelligent Virtual Environments. In: *Proceedings of the IEEE VR 2007*, 2007, S. 305–306

[Latoschik u. Fröhlich 2007b]
LATOSCHIK, Marc E. ; FRÖHLICH, Christian: Towards Intelligent VR: Multi-Layered Semantic Reflection for Intelligent Virtual Environments. In: *Proceedings of the Graphics and Applications GRAPP*, 2007, S. 249–259

[Latoschik u. a. 2006]
LATOSCHIK, Marc E. ; FRÖHLICH, Christian ; WENDLER, Alexander: Scene Synchronization in Close Coupled World Representations Using SCIVE. In: *IJVR* 5 (2006), Nr. 3, S. 47–52

[Latoschik u. Schilling 2003]
LATOSCHIK, Marc E. ; SCHILLING, Malte: Incorporating VR Databases into AI Knowledge Representations: A Framework for Intelligent Graphics Applications. In: *Proceedings of the 6th IASTED International Conference on Computers, Graphics and Imaging, CGI*, 2003

[Latoschik u. Tramberend 2012]
LATOSCHIK, M.E. ; TRAMBEREND, H.: A scala-based actor-entity architecture for intelligent interactive simulations. In: *Software Engineering and Architectures for Realtime Interactive Systems SEARIS, 2012 5th Workshop on*, 2012, S. 9–17

[Loock u. Schömer 2001]
LOOCK, Achim ; SCHÖMER, Elmar: A virtual environment for interactive

assembly simulation: From rigid bodies to deformable cables. In: *In 5th World Multiconference on Systemics, Cybernetics and Informatics*, 2001, S. 325–332

[Luck u. Aylett 2000]
Luck, Michael ; Aylett, Ruth: Applying Artificial Intelligence to Virtual Reality: Intelligent Virtual Environments. In: *Applied Artificial Intelligence* 14 (2000), Nr. 1, S. 3–32

[Lugrin u. Cavazza 2007]
Lugrin, Jean-Luc ; Cavazza, Marc: Making Sense of Virtual Environments: Action Representation, Grounding and Common Sense. In: *IUI*, 2007, S. 225–234

[Peters u. Shrobe 2003]
Peters, Stephen ; Shrobe, Howie: Using Semantic Networks for Knowledge Representation in an Intelligent Environment. In: *PerCom '03: 1st Annual IEEE International Conference on Pervasive Computing and Communications*. Ft. Worth, TX, USA : IEEE, March 2003

[Peuser u. Latoschik 2008]
Peuser, Nils ; Latoschik, Marc E.: Using Semantic Traversers to create persistent knowledge-based Virtual Reality applications. In: Schumann, Marco (Hrsg.) ; Kuhlen, Torsten (Hrsg.): *Virtuelle und Erweiterte Realität, 5. Workshop of the GI VR & AR special interest group*, Shaker Verlag, 2008 (Virtuelle und Erweiterte Realität, 5. Workshop of the GI VR & AR special interest group), S. 125–136

[Pfeiffer u. Wachsmuth 2008]
Pfeiffer, Thies ; Wachsmuth, Ipke: Social Presence: The Role of Interpersonal Distances in Affective Computer-Mediated Communication, CLEUP Cooperativa Libraria Universitaria Padova (Proceedings of the 11th International Workshop on Presence), 275–279

[Pfeiffer-Lessmann u. Wachsmuth 2008]
Pfeiffer-Lessmann, N. ; Wachsmuth, I.: Toward alignment with a vir-

tual human – achieving joint attention. In: *KI 2008: Advances in Artificial Intelligence*. Berlin, Germany : Springer, 2008, S. 292–299

[Quillan 1966]
QUILLAN, M R.: Semantic memory / DTIC Document. 1966. – Forschungsbericht

[Reiners u. a. 2002]
REINERS, Dirk ; VOSS, Gerrit ; BEHR, Johannes: OpenSG: Basic Concepts. In: *OpenSG 2002 Workshop, Darmstadt, Online*, 2002

[Sherman u. Craig 2002]
SHERMAN, William R. ; CRAIG, Alan B.: *Understanding Virtual Reality: Interface, Application, and Design*. Morgan Kaufman, 2002

[Soto u. Allongue 2002]
SOTO, Michel ; ALLONGUE, Sébastien: Modeling Methods for Reusable and Interoperable Virtual Entities in Multimedia Virtual Worlds. In: *Multimedia Tools Appl.* 16 (2002), Nr. 1-2, S. 161–177

[Sowa 2006]
SOWA, Timo: Towards the integration of shape-related information in 3-D gestures and speech. In: *Proceedings of the Eighth International Conference on Multimodal Interfaces*. New York : ACM Press, 2006, S. 92–99

[Sowa u. Wachsmuth 2005]
SOWA, Timo ; WACHSMUTH, Ipke: A Model for the Representation and Processing of Shape in Coverbal Iconic Gestures. In: OPWIS, K. (Hrsg.) ; PENNER, I.K. (Hrsg.): *Proceedings of KogWis05*. Basel : Schwabe Verlag, 2005, S. 183–188

[Stachowiak 1973]
STACHOWIAK, Herbert: Allgemeine Modelltheorie. (1973)

[Tramberend 1999]
TRAMBEREND, H.: AVANGO: A Distributed Virtual Reality Framework. In: *Proceedings Of the IEEE Virtual Reality*, 1999

[Wachsmuth 1994]
WACHSMUTH, Ipke ; CREMERS, Armin B. (Hrsg.) ; KAHLER, Helge (Hrsg.) ; LEHNER, Karin (Hrsg.): *Künstliche Intelligenz und Computergrafik: Abschlußbericht (Forschungsbereich III, KI-NRW Summary 93-94: 51-89).* 1994

[Wachsmuth 1998]
WACHSMUTH, Ipke: Das aktuelle Schlagwort: Virtuelle Realität. In: *Künstliche Intelligenz* 98 (1998), Nr. 1

[Wachsmuth u. a. 1995]
WACHSMUTH, Ipke ; LENZMANN, Britta ; JUNG, Bernhard: Communicating with Virtual Environments – A Survey of Recent Work at the University of Bielefeld. In: FELLNER, Dieter W. (Hrsg.): *Modeling - Virtual Worlds - Distributed Graphics MVD'95*, Infix-Verlag, 1995, S. 93–97

[WBS 2014]
WBS, AG: *www.techfak.uni-bielefeld.de/ags/wbski.* 2014. – Internetseite der AG-WBS; zuletzt besucht am 27.04.2014

[Wiebusch u. a. 2012]
WIEBUSCH, Dennis ; FISCHBACH, Martin ; LATOSCHIK, Marc E. ; TRAMBEREND, Henrik: Evaluating scala, actors and ontologies for intelligent realtime interactive systems. In: *Proceedings of the 18th ACM symposium on Virtual reality software and technology* ACM, 2012, S. 153–160

[Zachmann u. Rettig 2001]
ZACHMANN, Gabriel ; RETTIG, A.: Natural and Robust Interaction in Virtual Assembly Simulation. In: *Eighth ISPE International Conference on Concurrent Engineering: Research and Applications ISPE/CE*, 2001