Dynamische Steuerung von Avataren am Beispiel eines Theaterstücks

Diplomarbeit

Zur Erlangung des akademischen Grades

Diplom-Informatiker

an der

Fachhochschule für Technik und Wirtschaft Berlin

Fachbereich Wirtschaftswissenschaften II

Studiengang Angewandte Informatik

Schwerpunkt Multimedia

1.Betreuer: Herr Prof. Dr. Dipl.-Ing. Thomas Jung

2. Betreuer: Frau Prof. Dr. Elke Naumann

Eingereicht von

Steffen Rademacher am

18.11.2002

Inhaltsverzeichnis

1 Einführung

1.1 Vorwort

In der vorliegenden Arbeit sollen Möglichkeiten zur dynamischen, also frei definierten Bewegung von dreidimensionalen, interaktiven Inhalten, in Form einer Theaterstück-Simulation untersucht werden. Es werden verschiedene Technologien vorgestellt, um anhand der Anforderungen eine Wahl der zu benutzenden Technologie zu treffen. Für den Entwurf und die Implementierung eines funktionsfähigen Prototypen wird es notwendig werden, auch auf das Internet aufsetzende Hilfstechnologien zu benutzen. Um den Umfang der Arbeit jedoch überschaubar zu halten, werden diese Basistechnologien nur kurz erläutert.

1.2 Motivation

Seit der Entstehung der Computergrafik gibt es die Bestrebung, möglichst wirklichkeitsnahe Abbildungen der realen Welt mit Hilfe des Computers zu erschaffen. Im Zuge der Weiterentwicklung von zwei- zu dreidimensionaler Computergrafik und der kontinuierlichen Steigerung der Rechenkapazität von Computern, wurde ein Interesse an virtuellen Räumen und Realitäten in Echtzeit[1] geweckt.

[1] Alle Berechnungen zur Darstellung der dreidimensionalen Inhalte werden zur Laufzeit getätigt.

Der Rahmen der Entwicklung solcher Anwendungen wurde mit VRML, der Virtual Reality Modeling Language, erstmalig festgelegt. VRML, heute in der Version VRML97 (vormals auch VRML 2.0) vorliegend, stellt einen Standard der Realisierung virtueller Welten in Echtzeit dar, durch den eine Veröffentlichung im World Wide Web, sowie auch als eigenständige Anwendung, ermöglicht wird. Als weitere VR-Technologien sind Live3D von Netscape[2], Shout3D [3], Java3D[4], Pulse3D[5] und QuicktimeVR[6] zu nennen. Live3D und Shout3D basieren jeweils auf VRML, während Java3D, Pulse3D und QuicktimeVR eigenständige VR-Technologien sind.

Mittlerweile sind durch frei verfügbare Plug-Ins und spezielle Browser für die oben genannten VR-Technologien, Tausende von interaktiven, dreidimensionalen Welten im Internet verfügbar. Eine besondere Facette der Interaktivität ist es jedoch, wenn der Benutzer durch eine Personifizierung seiner selbst, oder durch das Steuern der Geschehnisse ein Bestandteil der Szenerie wird. Diese sogenannten Avatare finden bereits in einigen Systemen Verwendung. So gibt es interaktive Chats und Onlineshops, die Avatare implementieren, welche die Benutzer personifizieren oder unterstützen. Die vorliegende Arbeit untersucht, inwieweit ein interaktives Eingreifen innerhalb einer VR-Szene realisierbar ist, und welche Möglichkeiten bzw. Technologien dafür zur Verfügung stehen. Es wird weiterhin ein Prototyp zur Steuerung von dynamischen Inhalten innerhalb einer VR-Szene mit humanoiden Akteuren entwickelt. Dieser Prototyp soll die Möglichkeit bieten, ein virtuelles Theaterstück zu spielen.

[2] http://wp.netscape.com/eng/live3d/live3d_overview.html
[3] http://www.shout3d.com
[4] http://java.sun.com/products/java-media/3D/
[5] http://www.pulse3d.com
[6] http://www.apple.com/quicktime/qtvr/

2 Theaterstücke

Was macht ein Theaterstück aus? Auf welche Grundbausteine und Mechanismen kann es reduziert werden, um eine Umsetzung in ein VR-System zu ermöglichen? Der Autor hatte die Möglichkeit, am Berliner Gorki-Theater einen Einblick in die Arbeitsabläufe einer Theaterproduktion zu bekommen. Ein Theaterstück ist - rein technisch betrachtet - nur ein Zusammentreffen von Schauspielern, die eine Sammlung von Befehlen in einer Umgebung von Requisiten „interpretieren“ und „abspielen“. Natürlich wird diese These dem künstlerischen Anspruch einer Theaterinszenierung nicht gerecht, jedoch genügt sie aus der Sicht der Informatik, um daraus ein Abbild der Wirklichkeit in eine Computersimulation zu ermöglichen.

Eine Theaterproduktion ist ein iterativer Prozess: Für die Realisierung einer Inszenierung ist eine Vielzahl von Arbeitsschritten notwendig. So werden in zahlreichen Proben die Aktionen der Schauspieler präzisiert. Regisseur und Schauspieler bringen dabei ihre Vorstellungen meist gleichermaßen ein, und im Idealfall gelangen sie zu einem kooperativen Konsens. Ein Regieassistent ist dafür verantwortlich, dass die be- oder verarbeitete Dramenkonzeption aktuell verändert oder angepasst wird. Diese Änderungen werden in einem Drehbuch (manchmal auch Textbuch genannt) festgehalten.

2.1 Regieanweisung und Drehbuch

Regieanweisungen oder Drehbücher sind im Allgemeinen als ein Leitfaden für die an der Produktion eines Theaterstückes beteiligten Personen gedacht. Sie bilden also den Rahmen zur Produktion. Die modernen Inszenierungsauffassungen räumen den Akteuren auf und hinter der Bühne künstlerische Freiheiten ein, so dass objektiv nur eine Festlegung auf die genaue Einteilung in Akte und Szenen durch ein solches Drehbuch geschehen kann, bzw. die Regieanweisungen nur als Richtlinie gelten. Diese Interpretationen und Improvisationen sind, wenn überhaupt, jedoch schwer zu simulieren.
In [Bah00] kann eine Originalszene aus Schillers *„Kabale und Liebe“* gefunden werden, hier folgend einige Beispiele der Aktionsanweisungen (dramaturgische Hinweise) an die Schauspieler:

Akt II, 5. Szene - Schiller original

Ferdinand von Walter stürzt erschrocken und außer Atem ins Zimmer. Die Vorigen

FERDINAND War mein Vater da?

LUISE *(fährt mit Schrecken auf)*
Sein Vater! Allmächtiger Gott!

FRAU *(schlägt die Hände zusammen)*
Der Präsident! Es ist aus mit uns!

MILLER *(lacht voll Bosheit)*
Gottlob! Gottlob! Da haben wir ja die Bescherung!

FERDINAND *(eilt auf Luisen zu und drückt sie Stark in die Arme)*
Mein bist du, und würfen Höll und Himmel sich zwischen uns.

LUISE Mein Tod ist gewiß - Rede weiter - Du sprachst einen schrecklichen Namen aus - Dein Vater?

FERDINAND Nichts! Nichts! Es ist überstanden. Ich hab' dich ja wieder. Du hast mich ja wieder. o laß mich Atemschöpfen an deiner Brust! Es war eine schreckliche Stunde.

LUISE Welche? Du tötest mich!

FERDINAND *(tritt zurück und schaut sie bedeutend an)*
Eine Stunde, Luise, wo zwischen mein Herz und dich eine fremde Gestalt sich warf - wo meine Liebe vor meinem Gewissen erblaßte - wo meine Luise aufhörte, ihrem Ferdinand alles zu sein.

LUISE *(sinkt mit verhülltem Gesicht auf den Sessel nieder)*

FERDINAND *(geht schnell auf sie zu, bleibt sprachlos mit starrem Blick vor ihr stehen, dann verlässt er sie plötzlich in großer Bewegung)*
Nein! Nimmermehr!
[gekürzt]
(mit Entschluss auf sie zueilend)
Ich will sie führen vor des Weltrichters Thron, und ob meine Liebe Verbrechen ist, soll der Ewige sagen.
(er faßt sie bei der Hand und hebt sie vom Sessel)
[Rest wegen Umfang von 2 Seiten gekürzt]

Sind ein Großteil der Proben abgeschlossen, bekommt ein Inspizient das Textbuch. Die Aufgabe des Inspizienten ist es, die technische Umsetzung der

Aufführung hinter der Bühne von einem Kontrollpult aus zu steuern. Dazu werden Kontrollpunkte mit zugehörigen Aktionen koordiniert und in das Textbuch eingetragen. Daraufhin wird der richtige Einsatz an die jeweiligen Adressaten weitergegeben. So werden beispielsweise Akteure auf die Bühne geordert, die Lichttechniker angewiesen eine bestimmte Lichteinstellung vorzunehmen oder die Tontechniker veranlasst einen bestimmten Ton einzuspielen.
In [Köt01, S.49-51 u. S.64-66] sind original Drehbuchauszüge zu den Filmen *„Schlafes Bruder“* und *„Eyes Wide Shut“* zu finden. Grundsätzlich sind diese ähnlich zu dem Ausschnitt aus Schillers *„Kabale und Liebe“*, jedoch finden sich zusätzlich einige Anweisungen für Kamerabewegungen, insbesondere für Kamerawechsel und Totalen[1].

2.2 Analyse der Aktionsmöglichkeiten

Dem Ausschnitt von Kabale und Liebe zufolge, lassen sich stark vereinfacht, die Möglichkeiten der Schauspieler in einem Theaterstück zu agieren, auf zwei reduzieren: Bewegung und Sprache. Differenzierter kann man beide in weitere Kategorien unterteilen: Bewegung als Interaktion mit Requisiten und anderen Darstellern; Sprache in Form von Monolog oder Dialog mit Fokus auf die Intonation.
So eilt der Protagonist Ferdinand zu Beginn der Szene energisch auf die Bühne, auf der die Darsteller der letzten Szene zu finden sind, und eröffnet einen Dialog mit Luise. Luise (die in einem Sessel saß, und zu einem späteren Zeitpunkt in selbigem wieder Platz nimmt) springt auf und reagiert. Weitere Akteure (Vater und Mutter von Luise) beteiligen sich mit beiläufigen Bemerkungen an der Konversation.

[1] vgl. [Köt01, Einband] für Definitionen von Kameraeinstellungen

2.3 Vorbetrachtungen zum Prototyp

Um ein reales Theaterstück in eine virtuelle Form portieren zu können, müssen also folgende Rahmenbedingungen für die VR-Szene berücksichtigt werden:

- Interaktion
- Animation
- Bühnenaufbau
- Akteure (Schauspieler) und deren Kommunikation
- Schnittstelle zwischen Benutzer und System
- Schnittstelle zwischen System und VR-Szene

Eine Schnittstelle für die Umsetzung der Interaktion eines Benutzers mit dem System, sowie die Kommunikation des Systems mit einer VR-Szene, ist essentiell, da ein Benutzer letztendlich als Regisseur agiert. Er sollte das Stück nach seinen Vorstellungen, unter Berücksichtigung von formalen Regeln, zusammenstellen können.
Interaktion, z.B. in der Form von Kameraeinstellungen und -schwenks, ist insofern wichtig, als dass der Betrachter einer virtuellen Welt nicht direkt Teil dieser sein kann[2], wie im Gegensatz dazu ein Theaterbesucher (passiver) Teilnehmer an einer Inszenierung ist. Eine Kamera in der VR-Szene müsste einem Betrachter ermöglichen, die Welt aus verschiedenen Blickwinkeln erleben zu können. Inwieweit es dem Betrachter ermöglicht werden soll, diese Kamera selbst steuern zu können, muss dabei genauso diskutiert werden, wie die Anzahl und Art der vordefinierten Kameraeinstellungen auf die Szene und deren Objekte und Akteure.

[2] Ausnahmen sind hier hochpreisige Hardwaresysteme mit VR-Helmen und -handschuhen.

Animation ist ein weiterer essentieller Baustein für ein virtuelles Theaterstück. Insbesondere sind hierbei die Bewegungen der Akteure zu berücksichtigen, die ein relativ statisches Bühnen-Setup mit der benötigten Dynamik zu einem VR-Erlebnis machen müssen.

3 Technologische Grundlagen

3.1 Virtual Reality - Virtual Worlds

Eine greifbare Definition des Begriffes „Virtual Reality“ in der einschlägigen Literatur zu finden, erweist sich als schwierig. Eine allen Publikationen gemeinsame Aussage ist jedoch, dass es keine, aufgrund der Komplexität dieses Themas, wirklich allgemeingültige Definition gibt. Versuche, „Virtual Reality“ zu definieren und zu kategorisieren, zielen jedoch prinzipiell auf ein und dieselbe Aussage: *„Virtual Reality ist eine Möglichkeit für den Menschen, mit dem Computer zu kommunizieren und komplexe Daten zu visualisieren und zu manipulieren“* [Däß98, S. 101-102]. Wenn man sich die Übersetzung der Begriffe einmal genauer ansieht, wird der technisch-visuelle Aspekt sicherlich klarer. Virtual Reality, scheinbare, der Möglichkeit nach vorhandene Realität. In Anlehnung daran lässt sich für den Begriff „Virtual Worlds“ - übersetzt scheinbare Welten - eine anschauliche Definition herleiten: *„Virtuelle Welten sind computer-basierte Modelle eines dreidimensionalen Raumes und dessen Objekte mit begrenzten Interaktionmöglichkeiten. Ein Benutzer kann sich durch eine virtuelle Welt bewegen und mit deren Objekten in verschiedenen Wegen interagieren“* [Die01, S. 7-8].

An dieser Stelle wird auf Läge *„Skript Virtuelle Realität“* und Riegler *Virtual reality: Cognitive Foundations, Technological Issues & Philosophical Implications* verwiesen, wo das Thema „Virtual Reality“ ausführlich und sehr detailiert

diskutiert wird. Als Grundlage für ein echtes VR-System soll an dieser Stelle genauer auf die 3D-Computergrafik eingegangen werden.

3.2 3D-Computergrafik

Virtuelle Welten sind, wie festgestellt wurde, dreidimensionale Räume. Objekte werden im dreidimensionalen Raum durch Punkte dargestellt. Ein Punkt im Raum besitzt drei Koordinaten, x, y und z, welche die horizontale, vertikale und entfernungsbezogene Translation (Verschiebung) vom Koordinatenursprung [0,0,0] repräsentieren. Drei dieser Raumpunkte können den einfachsten dreidimensionalen Grundkörper aufspannen: Das Dreieck. Dies ist der Bestandteil aus dem die meisten Objekte der 3D-Computergrafik zusammengesetzt werden können. Natürlich können auch mehreckige Flächen durch eine größere Anzahl von Raumpunkten aufgespannt werden, jedoch werden, zumindest in der 3D-Computergrafik in Echtzeit, Vielecke zu Dreiecken reduziert. Dieses Konzept wird Tesselation genannt.

3.2.1 Polygone

Es gibt verschiedene Konzepte dreidimensionale Objekte zu erschaffen bzw. zu modellieren. Die herkömmlichste Methode ist die polygonale Modellierung. Jedes der gängigen 3D-Modellierungsprogramme bietet diese Möglichkeit an. Grundprinzip dabei ist, dass zwischen zwei Punkten (Vertices) eine Kante (Edge) verläuft, durch deren Verbindung Polygone entstehen. Mehrere dieser Polygone bilden dann eine Gitterstruktur, das sogenannte Mesh. Grundkörper wie Würfel, Zylinder, Kegel und Kugel sind in der Darstellung ein Mesh, dessen, durch Kanten definierte Flächen normalerweise gefüllt sind. Durch die Translation von einzelnen Punkten kann dann die Form der Objekte verändert und somit eine

neue Art von Körper geschaffen werden. Die meisten hochwertigen Modellierprogramme verfügen über vielseitige Werkzeuge zur Polygonbearbeitung. Auf diese Spezifika wird im Implementierungsteil dieser Arbeit eingegangen.

3.2.2 NURBS

Eine weitere Modellierungstechnologie sind NURBS (Non Uniform Rational B-Splines). Dabei handelt es sich um Geometrien, die durch Flächen mit unterschiedlicher Gewichtung an den Punkten durch die sie approximiert werden, beschrieben werden können. Durch die Variation der Gewichtung kann die Form verändert werden, ohne wie z.B. bei dem Polygonen die Punkte einzeln Transformieren zu müssen [vgl. Vol98]. In Abbildung 3.2.1 sind zwei instanziierte NURBS-Patches zu sehen, welche sich in der Gewichtung unterscheiden. Rechnerintern wird die Geometrie dann doch noch durch Tesselation zu einem polygonalem Mesh gerechnet, jedoch ist dieser Vorgang dynamisch, so dass ein Mesh verfeinert werden kann, wenn sich Parameter verändern (bspw. wenn der Betrachter näher kommt). Es sind so perfekt runde und komplexe Objekte möglich, die mit Polygonen nahezu unmöglich zu modellieren sind. Realistische, menschliche Gesichter sind ein Beispiel für eine Anwendung von NURBS-Patches (-flächen).

3.2.3 Subdivision Surfaces

Ein neueres Modellierungskonzept sind die Subdivision Surfaces. Als Basis zur Nutzung dieser Technologie kommen normale Polygonobjekte zum Einsatz, welche jedoch lokal an den gewünschten Stellen so unterteilt werden können (vgl. Abbildung 3.2.2), dass daraus Freiformen entstehen können. Einsatzmöglichkeit ist auch hier die Modellierung von Gesichtern, generell von Charakteren.

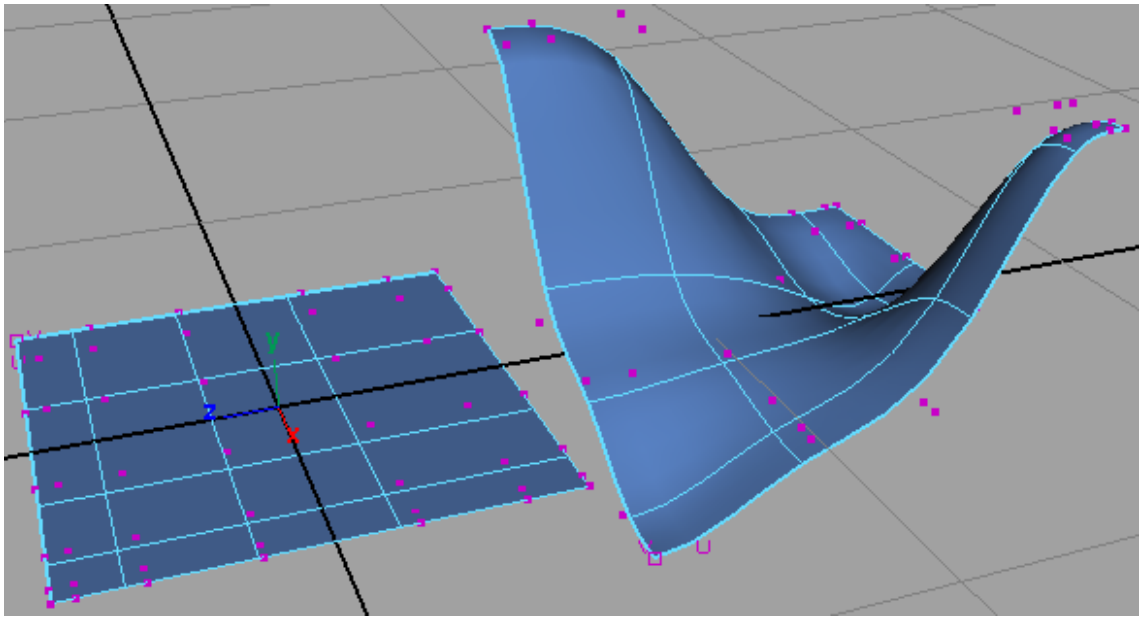

Abbildung 3.2.1: NURBS Patches mit unterschiedlicher Gewichtung

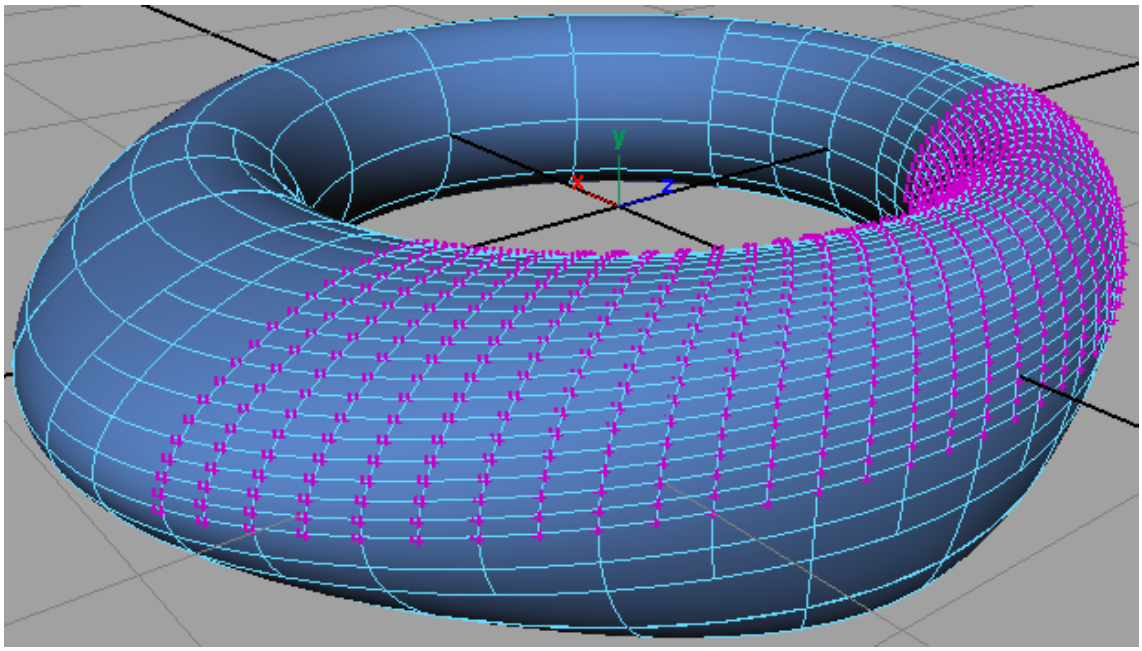

Abbildung 3.2.2: Subdivision Surface Torus mit vierfacher Verfeinerung

3.2.4 Engine

Die Darstellung von dreidimensionalen Daten benötigt grundsätzlich eine sogenannte Engine. Die Engine steuert die Berechnungen, die beim Transformieren, Rotieren oder Skalieren der 3D-Objekte notwendig sind und deren Darstellung in Echtzeit. 3D-Spiele, VR-Browser und -Plug-Ins sind Beispiele für Engines.

3.2.5 Produkte

Zur Modellierung gibt es eine Vielzahl von freien und kommerziellen Programmen. Die bekanntesten kommerziellen Produkte sind 3D-Studio Max von discreet[1], Maya von Alias/Wavefront[2] und Lightwave von Newtek[3]. Die Preise dieser professionellen Modellierprogramme liegen bei 2000-25000 Euro pro Lizenz[4]. Diesen Umstand haben sich engagierte Entwickler zu Nutze gemacht, und eine Vielzahl von günstigen oder sogar kostenlosen 3D-Entwicklungstools hergestellt. Im Allgemeinen kann man sagen, dass diese freien Programme inhaltlich und funktionell nicht an professionelle Produkte heranreichen. Besonders sind die Fähigkeiten zur Modellierung, der Animation und des Rendering, wenn überhaupt vorhanden, im Vergleich unterentwickelt. Jedoch nehmen sich Programme wie Spazz3D und CosmoWorlds Nischentechnologien wie VRML- und Avataranimation an, was das Bearbeiten dieses Projektes erheblich vereinfachen könnte.

[1] http://www.discreet.com

[2] http://www.aliaswavefront.com

[3] http://www.newtek.com

[4] Es gibt jedoch kostengünstige Studentenversionen oder in der Funktionalität eingeschränkte Versionen für den Privatgebrauch.

3.3 Avatare

Mit steigender Interaktivität im Internet wurde es obligat, dass VR-Anwendungen komplexer werden mussten. So wurde ein neues Anwendungsgebiet erschlossen: Avatare. Urspünglich stammt dieser Begriff aus der Theologie, genauer gesagt aus dem Hinduismus, wo der Avatar, unter der Vorstellung des Konzeptes der Wiedergeburt, den vergänglichen, menschlichen Körper bezeichnet. Mit der Entwicklung von Simulationen und Kriegsspielen der US Army in den 80er Jahren wurde der Begriff Avatar der allgemeinen Personifizierung eines Benutzers in einem beliebigen Objekt innerhalb einer virtuellen Umgebung bezeichnet [vgl. Has97, S.24-25]. Mit dem Fortschreiten der Computertechnologien und insbesondere der Computergrafik, hat sich im Laufe der Zeit eine andere Bedeutung herausgebildet: Ein Avatar ist demzufolge eine Kreatur (Tier, Monster, Mensch, Phantasiefigur), die von einem Benutzer gesteuert werden kann oder die selbstständig, anhand eines dynamisch generierten oder festgelegten Bewegungsablaufes, agiert [vgl. Däß98, S. 112].

3.4 VR-Technologien

Wie schon in der Einführung erwähnt, gibt es verschiedene Technologien für die Realisierung und Darstellung von VR-Inhalten. Diese Technologien müssen die, im Kapitel 2 genannten Möglichkeiten zur Darstellung von Bühne und Requisiten, Animationen, Aktionen und Interaktionen eines Theaterstücks bieten. Für die Umsetzung des Prototypen sollen diese, um eine Auswahl treffen zu können, kurz auf ihre Verwendbarkeit untersucht werden.

3.4.1 Live3D

Live3D baut auf der alten und statischen VRML 1.0 Spezifikation von 1995 [vgl. VRML 1.0] auf und unterstützt deshalb auch nicht die für das System essentiellen Anforderungen, wie z.B. Steuerung von Avataren, Animation und Interaktion in der Szene. Live3D implementiert jedoch eine Schnittstelle, welche die Manipulation statischer VRML 1.0 Objekte per Java/JavaScript in begrenztem Maße ermöglicht. Der Internetpräsenz des Herstellers Netscape lässt sich jedoch nicht entnehmen, ob die Software noch gepflegt oder überhaupt weiterentwickelt wird, was vermuten lässt, dass es auch in ferner Zukunft keine Version geben wird, welche die gewünschten Anforderungen im Rahmen des Systems erfüllen wird.

3.4.2 Quicktime VR

Quicktime VR ist im Gegensatz zu den anderen betrachteten Technologien nur eine „Pseudo 3D-Umgebung“, bei der zweidimensionale Bitmaps von, z.B. einer natürlichen Umgebung auf die Innenseite eines Zylinders gemappt[5] werden. Der Benutzer kann dann innerhalb dieses Zylinders die Kamera schwenken und hinein- oder herauszoomen. Schon allein der Fakt, dass Quicktime VR über keinerlei Möglichkeiten verfügt, echte dreidimensionale Objekte darzustellen bzw. in eine Szene zu integrieren, lässt es im Rahmen dieser Arbeit als Kandidaten ausscheiden.

3.4.3 Shout3D

Shout3D wiederum baut auf VRML97 auf und implementiert eine eigene Java/Javascript-Schnittstelle zum externen Steuern der Szenen mit speziellen, vordefinierten Methoden. Die Technologie unterstützt auch Avatare. Leider aber

[5] engl: planar aufgelegt

scheint das Interesse an Shout3D verlorengegangen zu sein, sodass auf der Firmenhomepage die letzten Neuerungen Mitte 2001 zu sehen waren. Des Weiteren kommen bei einer Nutzung noch Lizenzgebühren von 100 bis 200 US-Dollar[6] pro Jahr und Domain, auf welcher Shout3D-Inhalte zur Verfügung stellt werden, hinzu, was für ein freies und offenes Projekt schwer finanzierbar zu sein scheint.

3.4.4 Pulse3D

Pulse3D als weitere Internet-basierte VR-Technologie wurde inhaltlich und funktionell ebenfalls mehr oder weniger auf „Pseudo-3D“ reduziert. Der Pulse-Webseite wird die Technologie nun hauptsächlich für Online-Kommunikation, Marketing und E-Learning genutzt und weiterentwickelt [vgl. PULSE]. So ist es beispielsweise mit der Pulse3D-Lösung für Customer Support[7] möglich, kleine, dynamische Videos von (Verkaufs-)Beratern zur Verfügung zu stellen, die im Vergleich zu herkömmlichen Video-Streams höhere Qualität bei geringerer Dateigröße bieten. Dazu werden Bitmaps von Kopf und Gesicht eines solchen Beraters von der Software animiert. Die Software implementiert auch ein System zur Sprachgenerierung und zur phonembasierten[8] Lippenanimation. Auch wenn diese Funktionen beeindruckend sind, stellt Pulse3D nicht die benötigte Funktionstiefe zur Verfügung, um das angestrebte Projekt, inklusive einer beleuchteten VR-Welt mit Avataren, realisieren zu können.

3.4.5 Java3D

Java3D gilt im Allgemeinen als Technologie zum Programmieren von Stand-Alone Applikationen. Es ist aber auch durchaus möglich, Java3D-Inhalte, in Form von

[6] vgl. http://www.shout3d.com - „Buy Online Now“
[7] engl: Benutzerunterstützung
[8] Phonem - kleinste bedeutungsunterscheidende sprachliche Einheit.

Applets, online zur Verfügung zu stellen. Es gibt dazu verschiedene Loader, die beispielsweise VRML-Inhalte[9] in die Java3D-Umgebung laden. Blaxxun3D ist ein solcher Loader. Auch wenn aufgrund der Mächtigkeit von Java die Möglichkeiten als sehr vielfältig erscheinen, hat diese Technologie einen entscheidenden Nachteil: die Geschwindigkeit der Grafikengine. Mit durchschnittlicher Polygon- und Texturanzahl werden große VR-Welten auf einem Computer mit einem gängigen 1Ghz-Prozessor zu langsam dargestellt. Die Zahl, der in einer Sekunde darstellbaren Einzelbilder fällt unter 25 ab, was bei Bewegung der Inhalte für das menschliche Auge als eine stockende Wiedergabe wahrgenommen wird. Da die VR-Szene im Projekt hinsichtlich der Anzahl der Objekte und Polygone, als auch der Texturen, von einigem Umfang sein wird, muss von einer Umsetzung in Java3D abgesehen werden.

3.4.6 Shockwave3D

Das 3D-Format Shockwave3D von Macromedia Director ist eine der neueren VR-Entwicklungen. Im Gegensatz zu den zweidimensionalen Vorgängerprodukten kann Shockwave3D echte dreidimensionale Welten darstellen. Shockwave3D unterstützt den Import von 3D-Daten aus gängigen Softwareformaten, Partikelsysteme und Keyframe-Animation, Subdivision Surfaces, Mesh-Deformationen und Avatare. Eine Vielzahl von Medienformaten kann zum Erstellen von Shockwave3D-Projekten benutzt werden. Diese Funktionalitäten haben natürlich ihren Preis: 1200 US-Dollar muss für das Director 8.5 Shockwave Studio bezahlt werden.

[9] auch Avatare

3.4.7 VRML97

Als erster Standard zur Visualisierung Virtueller Welten manifestiert, kann VRML als die bedeutendste VR-Technologie bezeichnet werden. Wie den vorherigen Ausführungen zu entnehmen ist, haben sogar einige Entwickler anderer VR-Umgebungen VRML als Basis ihrer Entwicklung genommen und ihre Varianten entsprechend modifiziert und erweitert. Die Tatsache, dass VRML als offenes Projekt zu einem Großteil von der Internetgemeinde und von 3D-Computergrafik-Interessierten entwickelt und vorangetrieben wurde, kann als ein Zeichen anhaltenden Interesses gewertet werden, was eine Zukunft dieser Technologie sichert. Laut Sprachspezifikation besitzt VRML97 die notwendigen Mechanismen, die zu einer Umsetzung der in Abschnitt 2.3 genannten Anforderungen an das System nötig sind: Eine Programmierschnittstelle, die beliebige Erweiterbarkeit der Sprachelemente und die Möglichkeiten der Animation und Interaktion innerhalb einer Szene [vgl. Web3D A und Web3D B]. Desweiteren sind kostenlose Browser und Plugins von verschiedensten Herstellern netzweit verfügbar, bzw. schon bei Nutzern installiert.

3.4.8 Auswahl der VR-Technologie

Nach der Evaluierung der Leistungsfähigkeit der vorab aufgeführten Technologien, der vermutlichen zukünftigen Unterstützung und der Produkt- und Technologiepflege, wurde entschieden, für die Realisierung des VR-Systems VRML97 einzusetzen. Die Schnittstellen zwischen System und Benutzer erscheinen als ausgereift und gut implementiert und werden, den Herstellern zufolge, auch weiterhin kontinuierlich gepflegt und weiterentwickelt. Die - im Vergleich zu den genannten Technologien - weite Verbreitung von echten VRML-Inhalten und damit auch die der benötigten Browser und Plug-Ins im Internet legen den Grundstein für eine breite Erreichbarkeit und Akzeptanz des geplanten

VR-Systems zur dynamischen Steuerung von Avataren in einem Theaterstück. Der einzige ernsthafte Konkurrent zu VRML97 ist Shockwave3D. Jedoch sind Anschaffungspreis der Entwicklungsumgebung und die kommerzielle Natur dieser Technologie eindeutige Minuspunkte, die letztendlich VRML97 präferierten.

3.5 VRML

Um ein besseres Verständis für die gewählte Technologie zu bekommen, soll nachfolgend ein wenig genauer darauf eingegangen werden.

3.5.1 VRML Geschichte

Die ersten Ideen zu virtuellen Welten gab es, als nach Möglichkeiten der Realisierung einer dreidimensionalen, grafischen Benutzeroberfläche für das World Wide Web (WWW) gesucht wurde. Basis einer solchen Schnittstelle war eine Sprache zur Beschreibung von Objekten im Raum, sowie zu deren Darstellung und Verknüpfung im WWW. Auf der ersten internationalen Konferenz über das WWW im Genfer CERN im Mai 1994 wurde erstmals eine inoffizielle Sitzung mit dem Thema „Virtual Reality Markup Language and the World Wide Web“ abgehalten. Die Internet-Gemeinschaft wurde aufgerufen an der Entwicklung teilzunehmen, und bereits im Oktober 1994 konnte die Silicon Graphics, Inc. (SGI) VRML in der Version 1.0 präsentieren [vgl. Has97, S. 6].
VRML basiert im Wesentlichen auf der SGI Grafikprogrammierungsschnittstelle OpenInventor (OI), einer objektorientierten, auf OpenGL aufsetzenden API[10] zur 3D-Grafik Programmierung [vgl. Die97, S. 173]. OpenInventor-Quelltexte liegen im ASCII-Textformat vor. Das Dateiformat realisiert die vollständige Beschreibung

[10] Application Programming Interface

von 3D-Szenen mit Objekten, welche aus Polygonen zusammengesetzt sind. Des Weiteren sind Beleuchtung, Kameras, Materialien, Umgebungseigenschaften und interaktive Ereignisse Bestandteil des Formats. Eine Untergruppe des OpenInventor-Teams wurde beauftragt, OI für die Benutzung als Dateiformat für VRML umzugestalten. Im Zuge dieser Entwicklung wurden für VRML einige Funktionsmerkmale von OI, z.B. die Unterstützung von NURBS, reduziert. Die Entwicklung von VRML war damit jedoch nicht abgeschlossen. Zwei Jahre später, im August 1996, konnte die VAG[11] einen Entwurf zu VRML 2.0 vorstellen. Dieser Entwurf war das Resultat der Arbeit von beteiligten Firmen und Privatpersonen und erweiterte die Funktionalität von VRML um Interaktion (Sensoren), Animation (Interpolatoren) und eine Programmierschnittstelle (EAI - External Authoring Interface) [vgl. Has97, S6-7].

3.5.2 VRML Sprachelemente und Syntax

Wie alle Programmier- oder Skriptsprachen besitzt auch VRML formale Regeln. Diese Syntax besteht im Wesentlichen aus - in einem Szenenbaum hierarchisch angeordneten - Knoten mit Feldern und deren Feldwerten. Knoten sind Bausteine des VRML-Szenegraphen, die beispielsweise Geometrien, Materialien, Farben, Sensoren und Operationen auf Elementen einer Szene beschreiben [vgl. Däß99, S. 71-77]. Felder bestimmen Knotenattribute wie Größe, Radius oder Position. Die Feldwerte weisen diesen Attributen dann numerische Werte zu. Auf Geometrieknoten ausführbare Standardoperationen sind die Translation[12], Skalierung sowie die Rotation.

[11] VRML Architecture Group, erreichbar unter: http://www.web3d.org/fs_membersonly.htm; ein Zusammenschluss verschiedener Firmen zur Ausarbeitung des VRML-Standards, wie z.B. Intel, Microsoft und SUN.

[12] Engl: Verschiebung

Nachfolgend eine kurze Liste der Eigenschaften von VRML97 [vgl. Däß98, S.111].

- Eingebaute geometrische Primitive (Kegel, Kugel, Quader, Zylinder) einschließlich ihrer Flächendefinition
- Licht-, Material-, Textur- und Videosteuerung
- Räumliche Klänge (Wave- und Midi-Dateien)
- Animation, Ereignisbehandlung und -weiterleitung
- Kollisionserkennung
- Hyperlinks, Aussichtspunkte, Navigationsarten
- Prototyping zur Spracherweiterung
- Verschiedene Skriptsprachen für die Implementierung einer Szenenlogik

Animation und Prototyping werden nachfolgend genauer analysiert.

3.5.2.1 Animation in VRML97

Bewegungen in VRML97 können auf verschiedene Art und Weise realisiert werden. Unabhängig von der Methode liegt jedoch grundsätzlich derselbe Mechanismus zu Grunde. Da Animation eine Änderung der Position, der Rotation oder der Skalierung eines Objektes im Szenenbaum darstellt, ist sie nichts anderes als eine Anpassung der zugehörigen Koordinaten, Rotationswinkel und Skalierungsfaktoren - relativ zur Zeit - in der Szene. Angetrieben wird eine solche Bewegung durch einen `TimeSensor`, der kontinuierliche Abfolgen von Zeitwerten in Form von Ereignissen generiert und diese als Argumente an einen Interpolator-Knoten weitergibt. Diese Zeitwerte werden zur Interpolation der

Richtungsvektoren im Interpolator-Knoten verarbeitet, welcher wiederum Positionsereignisse an das zu bewegende Objekt sendet. Dort werden diese interpolierten Vektoren als neue Positionskoordinaten empfangen und das Objekt neu ausgerichtet [Has97, S. 72-73]. Dieser Mechanismus wird in folgender Abbildung schematisch dargestellt:

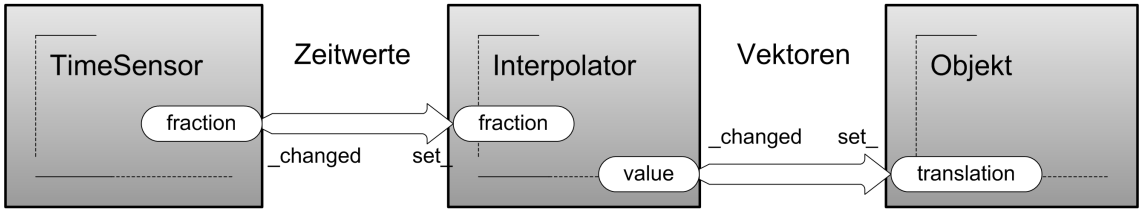

Abbildung 3.5.1: Route von Ereignissen bei einer VRML Animation

Sobald ein Zeitereignis vom `TimeSensor` ausgeht, wird dies als `faction_changed`-Ereignis an den zugehörigen Interpolator übergeben, welcher die aktuelle Zeit setzt (`set_fraction`) und anhand dieser Zwischenwerte aus den Schlüsselwerten interpoliert. Diese interpolierten Werte werden als `value_changed`-Ereignis an das Objekt geliefert, für welches per `set_translation`-Ereignis eine neue Position festgelegt wird. Der Interpolator wird vom TimeSensor solange Zeitereignisse erhalten, bis der Zyklus durchlaufen ist. Dieser Vorgang lässt sich natürlich per `loop`-Attribut beliebig oft wiederholen [vgl. Web3D A, Abschnitt 4.10].

Damit die erwähnten Elemente Ereignisse verschicken können, muss festgelegt werden welche Sensoren mit welchen Interpolatoren und Objekten verbunden sind. Dies wird in VRML97 durch das lineare Verschalten von den beteiligten Knoten erreicht [vgl. Web3D A, Abschnitt 4.10.2].

```
ROUTE TimeSensor.fraction_changed TO Interpolator.set_fraction
ROUTE Interpolator.value_changed TO Objekt.set_translation
```

Dieses Prinzip ist für alle Animationen in VRML97 gültig. Durch die Verwendung von knotenspezifischen Ereignissen können so auch Rotationen, Skalierungen, Aktivierung oder Deaktivierung von Knoten (z.B. Lichtquellen) gesteuert werden.

3.5.2.2 Prototypen

VRML an sich kennt nur eine begrenzte Anzahl von Knotentypen. Um komplexere Anwendungen realisieren zu können, wurde in VRML97 eine Möglichkeit implementiert, die Sprachsyntax zu erweitern. Diese Erweiterung wird Prototyp genannt. Ein Prototyp stellt im Prinzip eine abstrakte Vorlage dar. Durch das Festlegen von Ein- und Ausgabeereignissen und Feldern mit zugehörigen Feldwerten wird der neue Prototyp bestimmt. Der neu deklarierte Knoten ist beliebig wieder verwendbar, sowohl innerhalb der Szene, als auch in anderen Szenen [vgl. Web3D A, Abschnitt 4.8]. Bei letzterem muss der Prototyp jedoch als EXTERNPROTO per Inlineknoten[13] eingefügt werden [vgl. Has97, S.264]. Das allgemeine Gerüst eines Prototypen sieht folgendermaßen aus:

```
PROTO name  [
       #Deklarationsteil
                eventIn       Eventtyp  name
                eventOut      Eventtyp  name
                exposedField  Feldtyp   name   default
                field         Feldtyp   name   default
            ]
    {
       #Definitionsteil
       Knoten1 IS Knoten1 #oder Synonym
    }
```

Im Wesentlichen besteht ein Prototyp aus zwei Teilen, der Prototypdeklaration als Schnittstelle nach außen und der Prototypdefinition als Implementierungsteil. In dem Deklarationsteil werden alle Felder und Ereignisse angegeben, die später bei einer Instanziierung parametrisiert werden können, respektive als Ereignisse auf Instanzobjekten in Anwendung kommen können. Im Definitionsteil werden

[13] Inline importiert eine externe VRML-Datei in die Szene.

die formalen Felder des Deklarationsteils auf Konkrete abgebildet [vgl. Has97, S.261-262]. Der folgende Codeausschnitt zeigt einen Beispielprototyp aus dem Projekt:

```
PROTO Segment [
      exposedField   SFVec3f    centerOfMass  0 0 0
      exposedField   MFNode     children      []
      exposedField   SFString   name          ""
      exposedField   SFFloat    mass          0
      field          SFVec3f    bboxCenter    0 0 0
      field          SFVec3f    bboxSize     -1 -1 -1
     ]
  {
      Group {
            children IS children
            bboxCenter IS bboxCenter
            bboxSize IS bboxSize
            }
}
```

Dieses Codebeispiel stellt den Prototypen für die Segmentgeometrien eines H-Anim Avatars dar. Im Deklarationsteil werden alle benötigten Felder mit zugehörigen Datentypen (inklusive Standardwerten) deklariert: `centerOfMass` als SFVec3f (Single Field Vector mit drei Fließkommazahlen als Koordinaten), `children` als MFNode (Multi Field Node), `name` als SFString (Single Field String), `mass` als SFFloat (Single Field Float), `bboxcenter` und `bboxsize` jeweils als weitere SFVec3f. Im Definitionsteil werden dann Feldern, die veränderbar sein sollen, eindeutige Feldnamen zugewiesen. Im PROTO-Beispiel bleiben diese gleich (`bboxCenter IS bboxCenter`) und können so im gesamten VRML-Dokument eindeutig angesprochen und ggf. verändert werden.

3.6 Die H-Anim Spezifikation

Die Humanoid Animation Working Group wurde als eine Untergruppe des Web3D Konsortiums gegründet, um einen Standard für Humanoide in VRML97 zu manifestieren. Das heute in der Version 1.1 vorliegende Papier definiert

prinzipiell die Skelettspezifikation von Avataren. Demnach besteht ein solcher H-Anim Avatar aus einem System von Joints (Gelenken), an welche Segmente (Körperteile) gebunden sind. Die Gelenkdefinition ist ineinander verschachtelt. Vom Becken (Sacroiliac) gehen jeweils das linke und rechte Hüftgelenk (Hip) ab, überleitend zum Knie (Knee). An das Knie schließt sich das Sprunggelenk (Ankle) an. Den Abschluss der unteren Körperhälfte bilden die Füße, unterteilt in Subtalar, Midtarsal und Metatarsal[14] mit den anfügten Zehen. Vom Becken gehen die Joints des Oberkörpers ab. An das Becken sind Wirbelsäulengelenke angeschlossen, gefolgt von dem Joint `vc7`, welcher den Schultergürtel darstellt, also die Wurzel für die Schultergelenke mit anschließenden Ellenbogen-, Hand- und Fingergelenken. An `vc7` schließen ebenfalls die Hals- oder Genickgelenke mit angrenzender Schädelbasis (skullbase) an [vgl. HANIM, Abschnitt D.1-D.4]. Die H-Anim Spezifikation unterscheidet drei[15] verschiedene Detailstufen („Level of Articulation“), welche für unterschiedlich komplexe Skelette die korrekte Struktur der Gelenke definiert [vgl. HANIM, Abschnitt Appendix A]. So ist das komplexeste „LoA3“ eine anatomisch genaue Abbildung der Gelenkstruktur eines menschlichen Skelettes, was eine außerordentliche Genauigkeit bei der Modellierung ermöglicht. Die einfachste Form, ein LoA1-Skelett, umfasst die wichtigsten und grundlegenden Gelenke: Hüfte, Knie, Sprunggelenke, Schultern, Ellenbogen, Handgelenke und Genick. Damit kann ein einfacher, jedoch trotz reduzierter Gelenkanzahl beweglicher Humanoider geschaffen werden. Die Unterschiede zwischen den verschiedenen Detailstufen sind lediglich an den äußeren Extremitäten, den Händen und Füßen, sowie an der Wirbelsäule manifestiert. So werden bei einem LoA2-Skelett Hände und Füße um eingelenkige Finger bzw. Zehen erweitert. Das LoA3-Skelett definiert

[14] vgl. Anhang H-Anim Chart

[15] Genauer gesagt vier, jedoch kann das LoA0 vernachlässigt werden, da dort als einziges Gelenk die „humanoidRoot“ (vgl. Anhang H-Anim) definiert wird. Beim LoA0 handelt es sich somit nur um eine grundsätzliche Detailstufe.

Finger, Zehen und Wirbelsäule komplett anatomisch korrekt als dreigliedrige Extremitäten[16] [vgl. HANIM, Abschnitt Appendix A].

3.6.1 H-Anim spezifische Knoten

Für die Umsetzung der Spezifikation in VRML97, müssen einige Prototypen deklariert werden. Die folgenden Abschnitte beschreiben diese Knoten kurz.

3.6.1.1 Joint

Joints, also Gelenke, bilden das logische Skelett eines H-Anim Avatars. Im Gegensatz zu IK-Systemen[17], wie zum Beispiel dem von 3D-Studio Max, gibt es jedoch in einem H-Anim-Skelett keine Bones (Knochen). Die Joints werden im Prinzip als verschachtelte Punkte im Raum angelegt [vgl. HANIM97, Abschnitt B.1]. Bones hingegen sind schon eine visualisierte Version des logischen Skelettes, mit denen schon Animationen realisiert werden können, bevor überhaupt Segmentgeometrien erstellt und angebunden worden sind. Da Joints nur Punkte im Raum sind, lassen sich erst Bewegungen erstellen, sobald die Segmente angebunden sind. Man beachte das folgende Code-Beispiel:

```
DEF hanim_HumanoidRoot_0 Joint {
center  0 0.824 0.027
  children  DEF c_hanim_HumanoidRoot Group {
    children  [
      DEF hanim_sacroiliac_1 Joint {
        center  0 0.914 0.001
          children  DEF c_hanim_sacroiliac Group {
            children  [
              DEF hanim_l_hip_2 Joint {
                center  0.096 0.912 -0.001
                  children  DEF c_hanim_l_hip Group {
```

Die Joints werden per DEF-Befehl im gesamten VRML-Dokument bekannt gemacht, so dass sie eindeutig referenziert werden können. Mit dem Feld

[16] vgl. Anhang H-Anim

[17] Inverse Kinematik

`center` ist jeweils der zentrierte Raumpunkt des Gelenkes angegeben. Dem Codeausschnitt kann außerdem entnommen werden, dass vom Hauptgelenkknoten „HumanoidRoot“ ausgehend die untergeordneten Gelenke jeweils gruppiert werden. Das ist ein wichtiger Mechanismus, der bei der Animation von Körperteilen eine Rolle spielt: Wird beispielsweise der linke Arm am linken Schultergelenk (allg. Jointbezeichnung: `hanim_l_shoulder`) rotiert, müssen gleichzeitig alle hierarchisch untergeordneten Gelenke der Rotation folgen. Das betrifft das Ellenbogen-, Hand- und alle Fingergelenke (soweit vorhanden). Somit ist gewährleistet, dass diese Gelenke und folgend deren Geometrien (Segmente, vgl. Abschnitt 3.5.1.2) nicht einfach an Ort und Stelle „stehen bleiben“, sondern relativ mitgedreht werden, wie die folgende Abbildung illustriert:

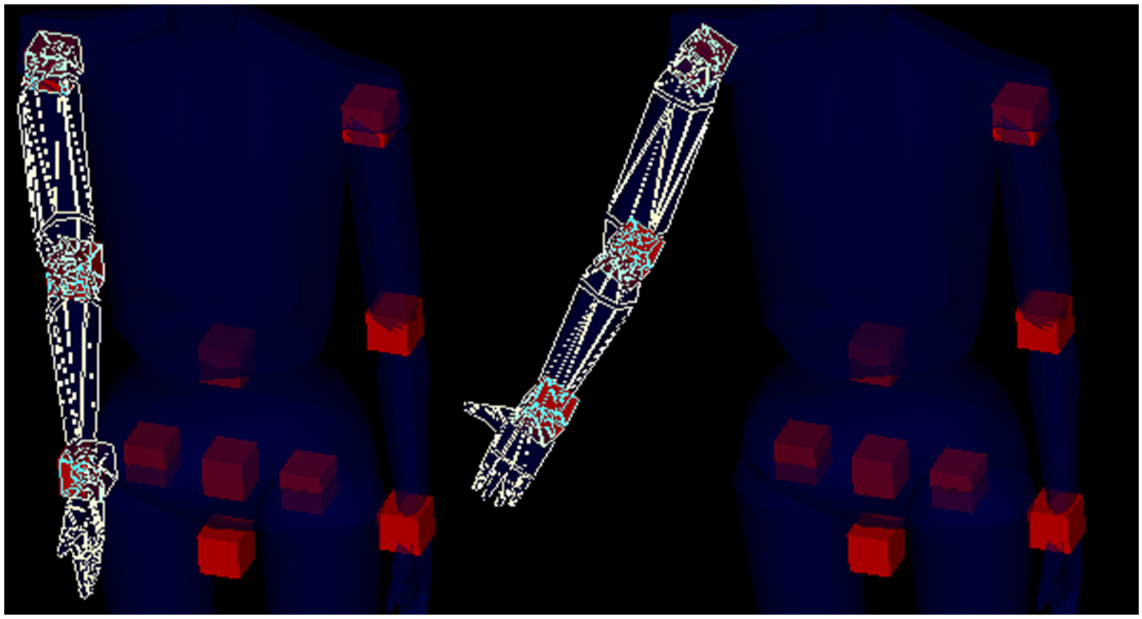

Abbildung 3.6.1: Rotation vom rechten Arm und untergeordneten Gelenken

3.6.1.2 Segment

Der Segmentknoten definiert die an die Jointstruktur angebundenen Körperteile eines H-Anim Avatars. Der Segmentknoten besitzt als Children typischerweise einen Shape- und Appearanceknoten [vgl. HANIM97, Abschnitt B.2]. Im Shapeknoten werden Geometrien, wie z.B. die Standardkörper Sphere, Box, Cylinder oder das komplexere IndexedFaceSet, welches eine Anzahl von Punkten im Raum über ein Netz von Verbindungen zu einer Oberfläche verbindet, definiert. IndexedFaceSet ist in VRML die Geometrie, die eine sehr genaue Modellierung von Objekten ermöglicht, und somit besonders für Avatare geeignet ist. Eine andere Vorgehensweise wäre, die Segmente als Inlineknoten in externe VRML Dateien auszulagern. Dieser Ansatz hat den Vorteil, dass Änderungen an den Segmenten nicht das Hauptdokument berühren, und dass Segmente leicht ausgetauscht und modifiziert werden können. Ein wenig problematisch ist jedoch die Anzahl an VRML Dateien, die durch das Auslagern entstehen - bei einem LoA3 Avatar etwa 70.

3.6.1.3 Site

Siteknoten dienen dem Anhängen von zusätzlichen Details wie Schmuck und ähnlichem an Joints bzw. Segmente. Jeder Jointknoten kann bis zu fünf Siteknoten besitzen [vgl. HANIM97, Abschnitt B.3]. Da es aber die Möglichkeit gibt, Schmuck und ähnliches auch durch Texturierung an eine Geometrie anzubringen, werden die Siteknoten normalerweise nicht genutzt. Das resultiert darüber hinaus in einer Einsparung an Quelltext von etwa 10% bei einem Level of Articulation 1 (LoA1) Avatar. Das entspräche rund 150 Zeilen VRML Quelltext.

3.7 Java

Im Jahre 1990 begann die amerikanische IT-Firma SUN Microsystems eine einfache Programmiersprache zu entwickeln, mit der Benutzeroberflächen für elektronische Unterhaltungs- und Haushaltsgeräte implementiert werden sollten. Diese Programmiersprache musste zuverlässig und aufgrund der schnelllebigen Architektur von Computerchips architekturunabhängig sein. Diese Eigenschaften ermöglichten eine unkomplizierte Portierung auf andere Systeme. Unter dem Namen „Oak“ entwickelt, wurde die Sprache wegen Urheberrechtsproblemen in „Java“ umbenannt. Als zu Beginn der neunziger Jahre die Bedeutung des Internets stetig anstieg, wurde erkannt, dass Java sich ausgesprochen gut zur Entwicklung von Internet-Applikationen eignete, welche plattformunabhängig auf verschiedensten Computer- und Betriebssystemen funktionstüchtig sein mussten. Seitdem wird Java von zahlreichen Internetbrowsern und Betriebssystemen unterstützt [vgl. Die97, S. 103].

Java bietet einen beträchtlichen aber überschaubaren Sprachumfang und ein objektorientiertes Sprachkonzept (Klassen, Objekte, Methoden und Vererbung). In Form von Applets können Java-Programme direkt in einem Webbrowser ausgeführt werden und dort interaktive Inhalte darstellen, oder Daten vom Browser oder der HTML-Umgebung verarbeiten [vgl. Die97, S. 104-105]. Um ein Java-Programm in einem Browser ausführen zu können, muss dieser Browser über eine Java Virtual Machine (JVM) verfügen. Die JVM ist ein Interpreter der den, durch das Kompilieren von Quelltext erzeugten Byte-Code eines Applets interpretiert und ausführt. Ist eine komplette Java-Entwicklungsumgebung (Java Runtime Environment, JRE) auf dem Computersystem installiert, übernimmt deren eigene JVM die Ausführung [vgl. Die97, S. 106].

3.8 WWW-Technologien

Es gibt zahlreiche Technologien, um die verschiedensten Inhalte im WWW darstellen zu können. Auf die für den Prototypen wichtigsten wird nachfolgend eingegangen.

3.8.1 HTML

HTML ist die Beschreibungssprache des Internets. Mit dem Kombinieren von layoutbeschreibenden Befehlen, den sog. Tags, können auf relativ einfach Art und Weise Internetseiten als Kombination von verschiedenen Medien, wie Bild, Text, Ton und Film, erstellt werden. Ein HTML-Dokument liegt im ASCII-Format vor, welches auf allen gängigen Betriebs- und Computersystemen verarbeitbar ist. Daraus leitet sich die wichtigste Eigenschaft von HTML ab: Es ist plattformunabhängig [vgl. Stu98, S.25]. Zur Darstellung von HTML-Inhalten wird ein Webbrowser benötigt, den es von verschiedenen Anbietern für die verschiedensten Plattformen kostenlos gibt. Ein herkömmliches HTML-Dokument ist statisch und repräsentiert nur die formatierte Darstellung von Informationen. Mit dem Einsatz von Formularen können diese statischen Dokumente durch Benutzerinteraktion dynamisiert werden [vgl. Stu98, S.193].

3.8.2 Formulare

Ein Formular in HTML implementiert Felder, Aktionsbuttons und Auswahlelemente zur Informationsverarbeitung. Typische Formularelemente sind Pull-Down-Menüs, Textfelder, Checkboxes sowie Radio-, Reset- und Submitbuttons [vgl. Stu98, S.194-205]. Ein Benutzer kann Textfelder mit Inhalten füllen und Optionen aus Menüs auswählen, welche dann bei einem Ausführen der verein-

barten Formularaktion von integrierten Funktionen verarbeitet, oder an externe Skripte übermittelt werden können [vgl. Stu98, S.193].

3.8.3 Dynamische Webseiten

Wann immer der Inhalt einer Webseite zum Zeitpunkt der Darstellung aktuell, interaktiv, personifiziert oder modular sein soll, kommen Technologien zum Generieren von dynamischen Webseiten zum Einsatz. Diese Technologien bauen solche Seiten anhand von vorprogrammierten Regeln auf und können Inhalte und Layoutvorschriften aus externen Daten[18] umsetzen. Zur Dynamisierung eingesetzte Technologien können in client- und serverseitige unterteilt werden. Der Unterschied beider ist, dass clientseitige direkt im Webbrowser des Benutzers, serverseitige jedoch nur auf dem Webserver, auf dem das HTML-Dokument verfügbar gemacht ist, ausgeführt werden können.

3.8.3.1 JavaScript

JavaScript ist eine clientseitige Skriptsprache zum Dynamisieren von Webseiten. Es ist der Standard zum Erweitern statischer Webseiten mit interaktiven Inhalten. Da JavaScript quasi der einzige clientseitige Standard für die Anreicherung von Webseiten mit dynamischen Elementen ist, ist es beinahe zu einem Bestandteil der HTML-Spezifikation geworden. Zusammengefasst wird dies unter der Bezeichnung DHTML[19]. JavaScript wird dabei direkt in ein HTML-Dokument eingebettet [vgl. Fla97, S.1-2].
Eine besondere Eigenschaft von JavaScript ist es, dass es sich hervorragend zum Manipulieren von HTML-Formularen eignet. So können die Werte von Formularelementen über Funktionen überprüft, verglichen oder manipuliert werden [vgl. Fla97, S.3-8].

[18] Beispielsweise aus einer Datenbank oder anderen Skripten.
[19] Dynamic HyperText Transport Protocol

3.8.3.2 PHP4

PHP ist eine serverseitige Skriptsprache, die, entweder in den HTML-Code eingebettet oder als eigenständiges Skript, HTML-Formatierungen ausgeben kann [vgl. Sch02, S.19]. In beiden Fällen werden die PHP-Befehle von einem Interpreter ausgeführt, der auf dem Webserver, der das angeforderte Dokument zur Verfügung stellt, läuft. Mit PHP können beispielsweise HTML-Formatierungen dynamisiert, Variablen zwischen Dokumenten weitergereicht oder Applikationen auf der Serverseite gestartet werden, um so Funktionen zu realisieren, die mit herkömmlichen HTML-Dokumenten nicht möglich wären [vgl. The00, S.11].

3.9 Schnittstellen

Als Schnittstelle zwischen Benutzer und VRML-Szene im Computer kommt ein VRML-Browser zum Einsatz. Diese Browser, als eigenständige Applikationen oder Plug-Ins für die gängigen Webbrowser verfügbar, interpretieren die Befehle und Beschreibungen in einem VRML-Dokument und ermöglichen eine Darstellung von VRML-Inhalten. Der Browser ist eine Engine.
Auch wenn VRML als standardisiert gilt, gibt es Unterschiede zwischen den verschiedenen VRML-Browsern und Plug-Ins. So gibt es auf der Microsoft Windows Plattform oftmals Probleme zwischen Betriebssystem, Internetbrowser und Plug-In, die dazu führen, dass Anwendungen nicht oder nur eingeschränkt funktionsfähig sind. In verschiedenen VRML-Browsern wie Cortona von Parallel Graphics, Contact von Blaxxun Interactive oder Cosmo Player von SGI/Cosmo Software kann es vorkommen, dass eine VRML-Welt unterschiedlich dargestellt wird und die integrierte Programmierschnittstelle die Funktion versagt.

3.9.1 External Authoring Interface

Das External Authoring Interface (EAI) ist die Programmierschnittstelle von VRML. Das EAI unterstützt JavaScript, ECMAscript und Java als Schnittstellensprachen. Die Benutzung dieser Programmier- bzw. Skriptsprachen eröffnet verschiedene Einsatzmöglichkeiten. Eine Anwendung kann von einfachen Schleifenprozeduren zum Füllen einer Szene mit Objekten bis zu vektoriellen Berechnungen zur Steuerung von Objekten innerhalb der Szene reichen. Die Programmiersprache Java kommt im Allgemeinen zum Einsatz, wenn ein Java-Applet diese komplizierteren Funktionen umsetzt oder eine grafische Benutzeroberfläche (Graphical User Interface - GUI) zur Steuerung eingesetzt werden soll. Laut VRML-Spezifikation muss zur Entwicklung von EAI-kompatiblen Programmen Java in der Version 1.1 zum Einsatz kommen. Zum Erstellen konformer Programme kann der Java Developer Kit[20] (JDK) in der Version 1.1.8 benutzt werden.

[20] Der Java Compiler, der installiert sein muss um Javaprogramme kompilieren zu können.

4 Konzeption und Design

Im Folgenden soll beschrieben werden, in welchen Schritten die Grundzüge des Systems entwickelt wurden.

4.1 3D-Modellierung

Ohne Zweifel muss der Visualisierung der Szenenobjekte eine große Bedeutung beigemessen werden. Letztendlich soll die Simulation, als virtuelle Repräsentation eines Theaterstückes, die entsprechenden optischen Effekte besitzen. Grundsätzlich besteht in Blaxxun Contact die Möglichkeit zu einer Darstellung von NURBS. Dies ist jedoch nicht konform zu der VRML97-Spezifikation, sondern eine eigenständige Erweiterung von Blaxxun Interactive. Es ist anzunehmen, dass die Darstellung der VRML-Welt in einem beliebigen anderen VRML-Browser nicht möglich ist, sobald Blaxxun-spezifische Knoten benutzt werden. So muss von der Modellierung der Avatare mit NURBS-Segmenten abgesehen werden, um eine weitgehende Kompatibilität von anderen VRML-Browsern zu gewährleisten. Zur Modellierung am besten geeignet scheint in diesem Falle die Software Maya 4.0 von Alias/Wavefront zu sein. Maya besitzt mächtige Werkzeuge zur Lowpoly-Modellierung[1] und verfügt darüber hinaus auch über einen VRML97-Exporter.

[1] Modellierung mit Fokus auf die optimalste (im Idealfall geringste) Anzahl von Polygonen.

4.2 VRML im Prototyp

Authentizität erhält die VRML-Szene, wenn neben den essentiellen 3D-Objekten, wie den Avataren, noch andere Objekte eingebunden werden. Um der Illusionästhetik einer Theateraufführung gerecht zu werden, ist eine kklar begrenzte Spielfläche, die Bühne, unabdingbar. Wird ein klassisches Stück inszeniert, könnten noch historische Bauwerke modelliert, und in der Szene platziert werden.
Beleuchtung, Nebel und andere bühnentechnische Effekte spielen athmosphärisch ebenso eine Rolle. Diese Punkte können während der Modellierungsphase näher untersucht, und gegebenenfalls nach Gutdünken implementiert werden.

4.2.1 Level of Articulation

In den Grundlagen zu der H-Anim Spezifikation wurde festgestellt, dass die Avatare in drei verschiedenen Detailstufen, den Level of Articulation, realisiert werden können. Überlegungen der Effizienz konnten bereits das detailreichste LoA 3 ausschließen, da die Anzahl der Dateien, die Menge der Daten und der Aufwand der Programmierung unnötig erscheinen. Es ist auch nicht zu vergessen, dass für jedes zusätzliche Gelenk eine weitere Geometrie, Textur und, was sich noch entscheidender auswirkt, ein zusätzlicher Interpolator mit zugehörigem Routing, für eine Animation innerhalb des VRML-Dokuments, zu implementieren ist. All das würde den Aufwand der Modellierung immens steigern, bzw. die Performance des Systems stark reduzieren. Der Kompromiss für eine Balance von Aufwand, Nutzen und Performance sieht eine Benutzung eines LoA 1 Skelettes vor. Bei diesem sind die 17 wichtigsten Joints und Segmente definiert und eine Bewegungsanimation dafür generiert maximal 18 Interpolatoren mit 36 Routen. Im Vergleich zu 75 Joints, 75 Segmentgeometrien

und 76 Interpolatoren mit 152 Routen für eine Bewegungsanimation sicherlich ein annehmbarer Kompromiss von Aufwand und Nutzen, der zudem schwächere Computersysteme nicht an die Auslastungsgrenze von Prozessor und Bandbreite bringt.

4.2.2 Einbindung der Segmentgeometrien

Da der Umfang der VRML-Szene überschaubar gehalten werden sollte, wurde überlegt, wie auf der einen Seite Zeilen von VRML-Code eingespart und auf der anderen Seite eine möglichst weitreichende Modularisierung der Szene erreicht werden kann. Diese Überlegungen haben zu der Idee geführt, dass die Segmente der Avatare als externe VRML-Dateien ausgelagert werden könnten. Durch diesen Schritt können diese unabhängig von der Hauptszene bearbeitet und bei Bedarf einfach ausgetauscht werden. Auch sollte so der Umfang des Hauptdokuments stark reduziert werden können: Auszugehen ist von einer Reduzierung des Hauptdokuments um vermutlich 4000 Zeilen VRML-Quelltext. Auf diese Weise können die Darsteller auf der virtuellen Bühne auch den unterschiedlichsten Ansprüchen der verschiedensten Stücke gerecht werden. Für ein klassisches Stück könnten die Segmente mit Texturen altertümlicher Kleidung bedeckt werden.
Sämtliche Requisiten könnten zudem auf die gleiche Weise, als externe Dateien vorliegend, unabhängig von der Hauptszene einzeln modifiziert und bei Bedarf integriert werden.

4.2.3 Bewegungsanimationen

Der Vorteil der Benutzung der H-Anim konformen Avatare ist, dass die Gelenkstruktur aller H-Anim Avatare kohärent ist. So können die Bewegungsanimationen, soweit sie vollständig vorliegen, für andere H-Anim Avatare wieder

verwendet werden, bzw. andere, öffentlich verfügbare, Bewegungsanimationen benutzt werden. Dies wird einerseits dadurch erreicht, dass Rotationen von Segmenten nicht absolut, sondern relativ zur Position des korrelierenden Joints ausgeführt werden, und andererseits die Position der Joints durch das H-Anim-Papier standardisiert ist [vgl. HANIM, Appendix A].

Um zunächst einen Überblick der Anzahl und Art der benötigten Animationen zu bekommen, müsste eine Szene näher betrachtet werden. Der Gedanke war, dass es zu jedem Theaterstück eine Regiefassung geben muss, in der sehr genau und explizit festgehalten ist, welche Aktionen ein Schauspieler in einer bestimmten Raum-Zeit-Sequenz ausübt. Diese Theorie war mit dem Gedanken verbunden, dass nicht jeder Schauspieler unbegrenzt Vorbereitung auf ein einzelnes Stück haben kann, und deshalb schon eine relativ genau vordefinierte Handlungsabfolge vorliegt.

In Abschnitt 2.1 wurde bereits ein Ausschnitt des Sturm und Drang-Dramas „Kabale und Liebe“ (1783) von Friedrich Schiller vorgestellt. Aus den dramaturgischen Hinweisen für die Akteure, die dem Text vorangestellt sind, lassen sich Bewegungen ableiten. So kann *„stürzt betäubt zu Luisens Füßen nieder“* als eine „Ohnmachts“-Animation umgesetzt werden, bei der ein Avatar sich benommen mit der Hand an den Kopf fasst, um dann taumelnd zu Boden zu gehen. Dem folgend muss es eine Bewegungsanimation geben, bei der dieser Avatar dann wieder aufsteht, um in die Ausgangspose zurückzukehren. Auch ist im gleichen Zug eine „Idle“-Animation zu integrieren, die es ermöglicht, den „ohnmächtigen“ Avatar eine beliebige Anzahl von Animationszyklen regungslos am Boden liegen zu lassen. Allgemein können die Bewegungsanimationen somit in zwei Gruppen unterteilt werden:

- einzelne, unabhängige und
- zwei- oder mehrteilige

Bewegungen, die keine nachfolgenden bedingen, wären z.B. „Verbeugen“, „(Ab)Winken“, „Bejahen“ und „Verneinen“, „Ohrfeigen“ und ähnliche. Mehrteilige könnten folgende sein: „Kniefall mit Aufstehen“, „Laufen zu Punkt X und zum Ausgangspunkt zurückkehren“, „Umarmen“.

4.3 Die Steuerungslogik

Die Steuerunglogik realisiert die eindeutige Zuordnung von Aktionen und Befehlen. Befehle werden Akteuren zugewiesen, die diese dann umsetzen.

4.3.1 Aufbau der Regieanweisungen

In den vorangehenden Ausführungen wurden bereits die Eigenschaften von Regieanweisungen und von Theaterstücken knapp skizziert. Für das VR-System müssen die Anweisungen jedoch expliziter sein, da Improvisation und Interpretation im System nicht möglich sind. Die Regieanweisungen sind also mehr als konkreter Befehl denn als Richtlinie zu verstehen. Mit diesen Erkenntnissen lassen sich Überlegungen über ein einfaches, erweiterbares Sprachformat anstellen. Dieses lässt sich in einer linearen Blockform mit der Semantik `Elementbezeichner:Element` organisieren. Bezeichner und Element werden durch einen Doppelpunkt als `Delimiter`[2] voneinander getrennt, während die gesamten Blöcke von einem waagerechten Strich abgegrenzt werden. Elementbezeichner repräsentiert das in der VRML-Szene angesprochene Objekt, wie Avatar, Kamera oder Text. Element ist in diesem Zusammenhang die Änderung der Bewegungsanimation, Kameraposition bzw. des Textinhaltes. Somit lassen sich zu verändernde Elemente, eine Definition vorausgesetzt, eindeutig referenzieren. Innerhalb der VRML-Szene wird dann genau festgelegt,

[2] Ein festgelegtes Zeichen zum Trennen von Zeichenketten.

welche Anweisung auszuführen ist. Weiterhin müssten diese Befehle beliebig erweiterbar sein, um so zusätzliche Funktionalitäten, welche bei einer Weiterentwicklung des Systems über die Arbeit hinaus notwendig werden könnten, integrieren zu können.

4.3.2 Anweisungselemente

Geplant ist, die Avatare und Kamerapositionen innerhalb der Szene zu bewegen, sowie den angezeigten Text zu modifizieren. Der Gedanke diese Anweisungen in linearer Blockform zu arrangieren, resultiert in folgender Elementdefinition.

4.3.2.1 Bewegungsanimation

Eine Möglichkeit Linearität und Modularität einer Anweisung zum Abspielen einer Animation zu erreichen, ist den zu bewegenden Avatar mit der zu spielenden Animation in einem String zu verknüpfen:

```
avatar1:anim1|avatar1:anim5|avatar1:anim4|
```

Um die Syntax einfach zu halten, werden die Bewegungsanimationen als `animX` definiert. Im VRML-Quelltext können somit Animationen je nach Anspruch des Stückes einfach ausgetauscht werden. Innerhalb des Frontends, welches die Bezeichnungen der Animationen beinhalten muss, und des VRML-Quelltextes (TimeSensor mit Verbindung zu den Interpolatoren) ist jedoch eindeutig definiert, um welche Bewegung es sich bei `anim3` handelt (bspw. „inohnmachtfallen“). So würde ein Element der Form `avatar1:anim3` in der Szene in einem „ohnmächtig umfallenden“ Avatar Nummer 1 resultieren.
Um einen zweiten Avatar in derselben Szene zur gleichen Zeit zu steuern, bedarf es eines zweiten Anweisungsblock-Strings. Dieser liegt in der gleichen Syntax

vor, jedoch unterscheiden sich natürlich die Elementbezeichnung (`avatar2`) und ggf. die Elemente.

```
avatar2:anim3|avatar2:anim2|avatar2:anim8|
```

Sollten zusätzliche Avatare in die Szene implementiert werden, können diese durch die Benutzung von Anweisungsblöcken in der Form `avatarX:animX` gesteuert werden.

4.3.2.2 Textänderungen

Das Modifizieren der angezeigten Texte der Avatare ist ebenfalls als wichtiger Bestandteil in die Regieanweisungen zu integrieren. Den Überlegungen zu der Implementation von Kommunikationsmitteln in der VRML-Szene zufolge, sollen die Text-Knoten in VRML als MFString (Multi Field String, grundsätzlich ein Stringarray) mit drei Textzeilen realisiert werden. Das Webfrontend stellt dazu drei separate Eingabefelder für Eingaben zur Verfügung. Diese drei einzelnen Textzeilen sind eindeutig einem Textobjekt, und somit einem Avatar, zugewiesen.

```
avatartext1:Hallo;Wie geht;es dir?;|avatartext2:Hey;Mir gehts gut;;|
```

Mit der Trennung der Zeilen durch Semikola können diese jeweils an das entsprechende Feld des Stringarrays weitergegeben werden. Sind in einem Teil des Textblocks keine Zeichen, muss eine leere Zeile dargestellt werden.
Da der Text in unmittelbarer Beziehung zu einem speziellen Avatar steht, wäre es vorteilhaft, die Animationselemente mit den Textelementen zu verbinden. Aufgrund der Modularität der Befehlssyntax ließen sich beide Elemente in folgender Art und Weise verbinden:
So sind Text und Animation in einem Block miteinander verbunden, jedoch für die einzelnen Avatare separiert.

```
avatar1:anim2|avatartext1:Hallo;Wie geht ;es dir?;|

avatar2:anim5|avatartext2:Hey;Mir gehts gut;Und dir?;|

avatarX:anim8|avatartextX:Aber Hallo!;Mich gibts auch noch;;|
```

4.3.2.3 Kameraänderungen

Eine Änderung der Kameraposition zwischen allen möglichen, in der Szene vordefinierten Blickpunkten (Viewpoints), ist ebenfalls als Teil einer skriptbasierten Steuerung zu betrachten. Es bietet sich an, diese Anweisungen, von der Avatarsteuerung getrennt, als separaten String zu übergeben.

```
view:1|view:3|view:8|view:5|view:3|view:6|view:7|
```

Durch ein Anordnen der möglichen Viewpoints in einer Liste, können diese mit einer einfachen Nummernangabe referenziert werden. Das `view`-Tag beschreibt lediglich, dass es sich bei dem Inhalt des Strings um die Reihenfolge der einzustellenden Kamerapositionen handelt.

4.3.3 Webfrontend

Mit der Definition der Befehlsstrukturen lässt sich eine Schnittstelle zum Erstellen der Abläufe eines virtuellen Theaterstücks für den Benutzer entwickeln. Diese Benutzeroberfläche kann optimaler Weise in HTML umgesetzt werden. Durch die Benutzung von einem Formular mit entsprechenden Elementen und hinterliegenden Funktionen können die folgenden Anforderungen erfüllt werden:

- Zusammenstellen der Bewegungsabfolgen für die Avatare
- Darzustellende Texte
- Veränderung der Kameraposition

- Titel, Autor und zusätzliche Informationen zum Stück

Es lassen sich die in Abbildung 4.3.1 dargestellten USE-CASES definieren.

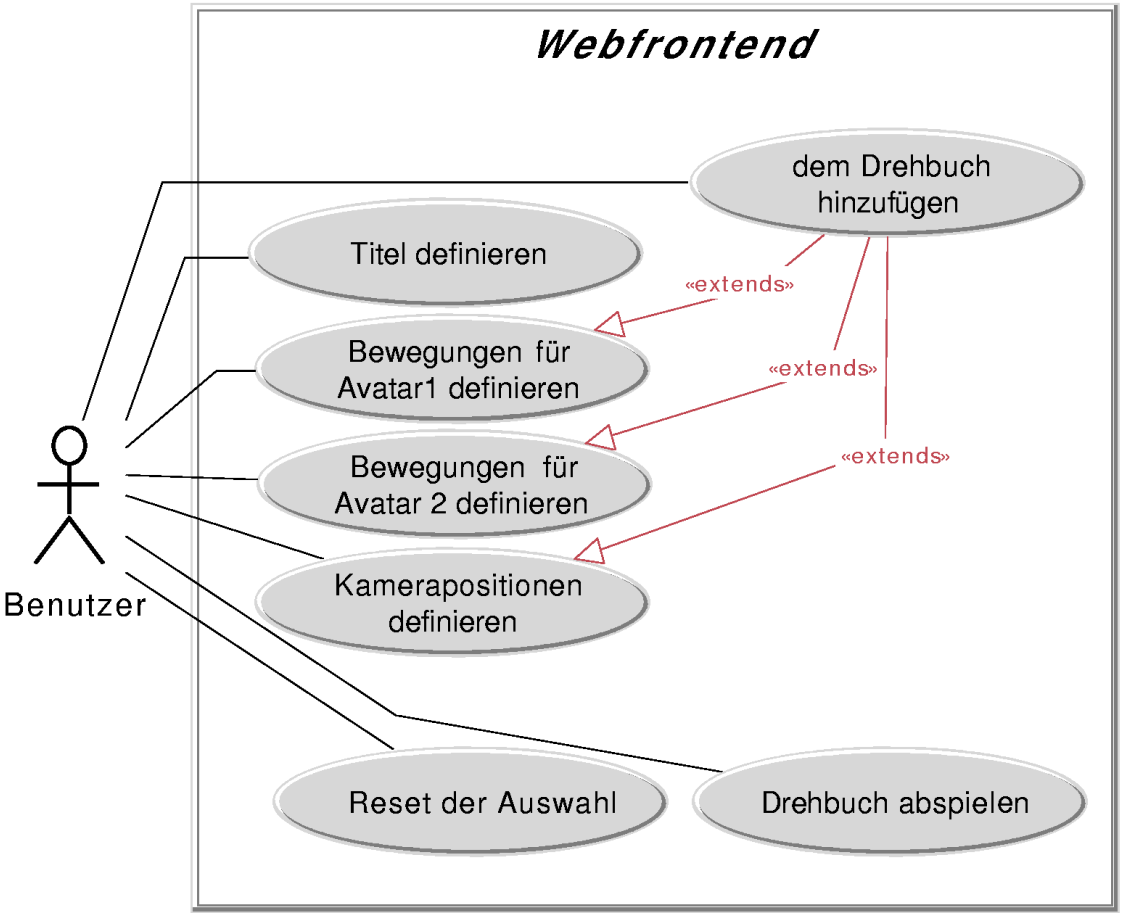

Abbildung 4.3.1: USE-CASE Diagramm

Es bietet sich an, die verfügbaren Bewegungsanimationen sowie die Kamerapositionen jeweils in einem Pull-Down-Menü anzuordnen. So kann der Benutzer einfach die zu spielende Animation oder aktive Kameraeinstellung auswählen. Nach Auswahl dieser, können die Textzeilen der Avatare ausgefüllt werden. Bei der Betätigung eines „Hinzufügen“-Buttons können Auswahl und Eingabe jeweils in einem Blockteil zwischengespeichert werden. Um nachvollziehen zu können,

welche Blöcke bereits vorbereitet worden sind, werden diese komplett in Statusfeldern ausgegeben. In diesen Feldern könnten dann noch jeweils per Hand Änderungen vorgenommen werden. Die folgende Abbildung stellt einen Entwurf dieses Webfrontends vor:

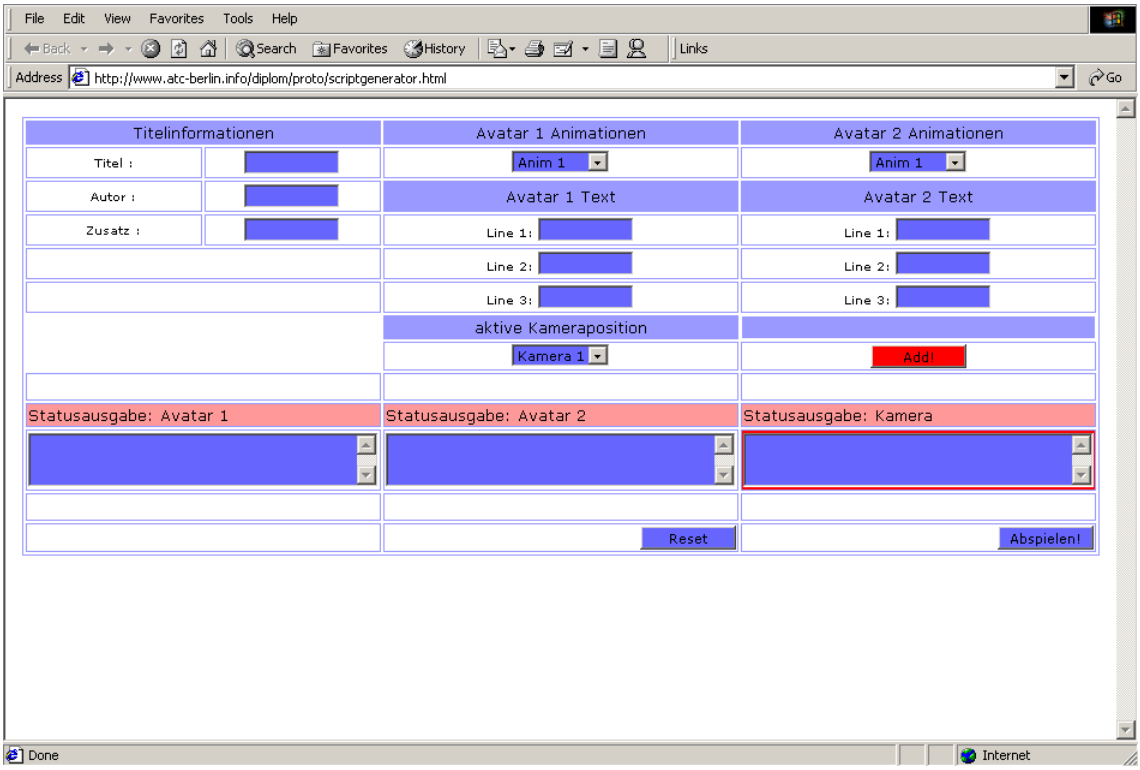

Abbildung 4.3.2: Webfrontend

Aus der Abbildung ist zu entnehmen, dass Einstellungsmöglichkeiten für zwei Avatare vorgesehen sind. Im oberen Teil können die Bewegungsanimationen aus den Optionen des Pull-Down-Menüs ausgewählt werden. Darunter sind die Textfelder für den anzuzeigenden Text platziert. Daran anschließend ist das Menü für die Kameraposition arrangiert. Bei Betätigung des „Hinzufügen!“-Buttons wird die jeweilige Auswahl als neuer Block den gesamten Anweisungen angehängt. Bevor die Zusammenstellung eines Stückes abgeschlossen ist, können in den linken Textfeldern auch noch nähere Angaben zum Titel und

Autor gemacht werden. Diese können dann vor einem Abspielen des Stückes angezeigt werden[3].
Wird der „Abspielen“-Button betätigt, werden die zusammengestellten Anweisungen zur weiteren Verarbeitung an die EAI-Steuerung übergegeben.

4.4 Einsatz des EAI zur Steuerung

Die Vorüberlegungen zur Steuerung der VRML-Szene haben sich mit dem EAI, dem External Authoring Interface, auseinandergesetzt. Da das EAI Java, JavaScript sowie ECMAscript als Schnittstellensprachen zulässt, wurde die Überlegung angestellt, welche der Sprachen am geeignetsten sei. Da es bei Javascript teilweise schwerwiegende Implementierungsunterschiede und Funktionsprobleme zwischen den verschiedenen VRML- und Webbrowsern gibt, wurde von der Benutzung für die Steuerung abgesehen. Die Benutzung von ECMAscript stellt sich im gleichen Zuge als hinfällig dar, da es es sich dabei lediglich um eine, in unserem Falle, VRML-spezifische Variante von Javascript handelt, und nur innerhalb des VRML-Dokuments eingesetzt werden kann, um Ereignisse intern zu bearbeiten oder an die Javascriptumgebung des umrahmenden HTML-Dokuments weiterzugeben. Die Umsetzung des entwickelten Interfaces lässt sich so nur wirklich effektiv mit Java realisieren. Das EAI unterstützt nur Java in der Version 1.1, so dass das „neuste“, unterstützte Java 1.1 JDK der Version 1.1.8 zum Einsatz kommen muss.

4.4.1 Funktionalitäten

In den vorangegangenen Abschnitten haben sich schon einige Rahmenbedingungen abgezeichnet. So werden Methoden für die Manipulation der Szenen-

[3] Z.B. Titel: Romeo und Julia, Autor: Sharespeare, Inszeniert von: Chr. Schlingensief

elemente für Text, Kameraposition und Bewegungsanimation ebenso benötigt, wie Methoden zum Erstellen, Analysieren, Vorbereiten und Abarbeiten der vom Benutzer erteilten Anweisungen. Weiterhin muss eine Übertragung dieser Informationen übernommen werden. Letztlich wird auch noch eine Benutzeroberfläche benötigt, die mit einfachen Bedienelementen das Abspielen des Stücks ermöglicht und auch systemrelevante Informationen ausgeben kann.

4.4.2 Userinterface

Da ein Benutzer hauptsächlich in den Formularen des Frontends die Reihenfolge und Art der Interaktionen in der VRML-Szene festlegt, sollten die Interaktionsmöglichkeiten im Applet auf das Wesentliche reduziert werden. Wie in der Abbildung zu sehen ist, kann die grafische Oberfläche (GUI) in einen Bedienteil auf der rechten Seite, und in ein Statusfenster auf der linken Seite unterteilt werden (Abbildung 4.4.1). Im Bedienteil sind die „Play“, „Pause/Resume“, „Stop“

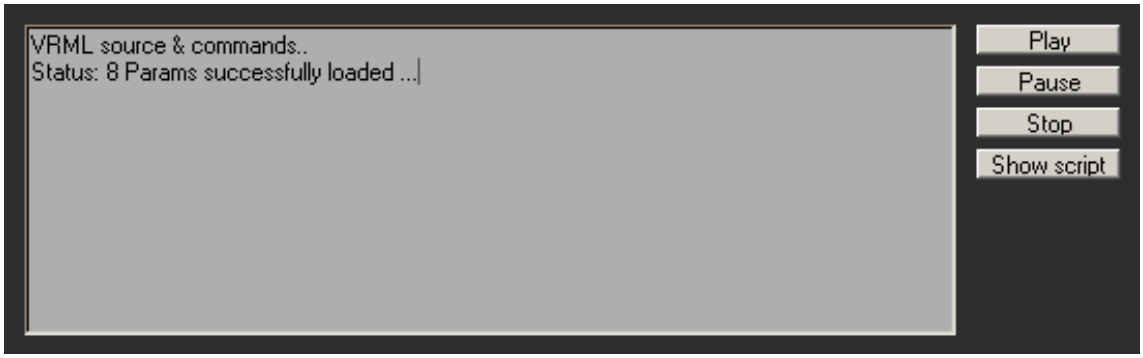

Abbildung 4.4.1: Applet-GUI

und „Show Script“-Buttons platziert. Bei Betätigung des „Play-Buttons“ sollen die vorher im Frontend zusammengestellten Aktionen in der VRML-Szene ausgeführt werden. Der „Pause-Button“ unterbricht diese Wiedergabe und wird

nach Betätigung in einen „Resume-Button“ umgewandelt[4], welcher bei erneuter Betätigung das Abspielen an der pausierten Stelle fortsetzt. Der „Stop-Button“ beendet das Abspielen des Stückes. Ein Klicken auf den „Show Script-Button“ löst die Ausgabe der abzuspielenden Aktionen im Statusfenster aus. Das Statusfenster informiert den Benutzer über die Anzahl der zu spielenden Animationen, darüber welche Animation im Moment gespielt wird, die aktuelle Kamera und die Textausgabe der Avatare und ist auch als Debug-Umgebung, in der die steuerungsrelevanten Informationen zur Laufzeit ausgegeben werden können, gedacht.

4.4.3 Kommunikation

Das Kommunikationskonzept lässt sich in zwei Teilen einrichten. Im ersten Teil wird die vom Benutzer im Webfrontend getroffene Auswahl an Aktionen an die EAI-Steuerung weitergegeben und für die Manipulation der VRML-Szene vorbereitet. Der zweite Teil ist die Weitergabe der verarbeiteten Befehlsobjekte von der EAI-Steuerung an die VRML-Szene und deren Knoten.
Da VRML-Szene und EAI-Steuerung in einem Browserfenster laufen müssen, kann die Übermittlung der Steuerungsstrings über das HTML-Dokument, welches beide Komponenten integriert, geschehen. Die EAI-Steuerung als Applet kann diese Parameter dann zur weiteren Verarbeitung abgreifen. Nach einer Verarbeitung der Anweisungen in ausführbare Befehle müssen diese dann von dem Applet an die VRML-Szene zur Manipulation der Knoteninhalte weitergegeben werden, wie folgende Abbildung illustriert.

[4] Beschriftung und Funktion werden dafür modifiziert.

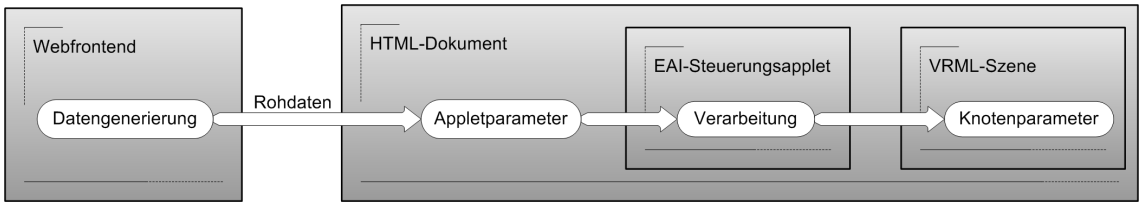

Abbildung 4.4.2: Kommunikation im Prototyp

4.4.4 Benötigte Klassen

Im Sinne des OOP-Ansatzes[5] ist eine Auslagerung der verschiedenen Funktionen zur Manipulation der Szene laut Abbildung 4.4.3 in eigenständige Klassen vorzunehmen. Da die Anweisung dynamischer Natur sind, können auf diese Art und Weise unabhängige Objekte zur Steuerung benutzt werden.

4.4.5 Manipulation der Szene

Es besteht die Möglichkeit, via EAI alle VRML-Knoten innerhalb eines Dokumentes anzusteuern, solange diese Knoten über Ein- und Ausgangsschnittstellen verfügen. Im Allgemeinen sind solche EventIns und EventOuts als Attributfelder schon definiert. Falls doch ein Knoten nicht mit diesen Feldern ausgestattet sein sollte, können per Prototypdefinition und -deklaration die benötigten Felder hinzugefügt werden. Nun implementieren die, für das System interessanten Knoten, Text, TimeSensor und Viewpoint, diese Schnittstellen schon.
Unter der Benutzung der Objekte der benötigten Klassen, können die Knoten in der VRML-Szene mit den notwendigen Informationen zur Manipulation versorgt werden. In Abbildung 4.4.4 ist ein Sequenzdiagramm zur Vorbereitung und Verarbeitung der Animationsinformationen dargestellt.

[5]Objekt-Orientierte Programmierung

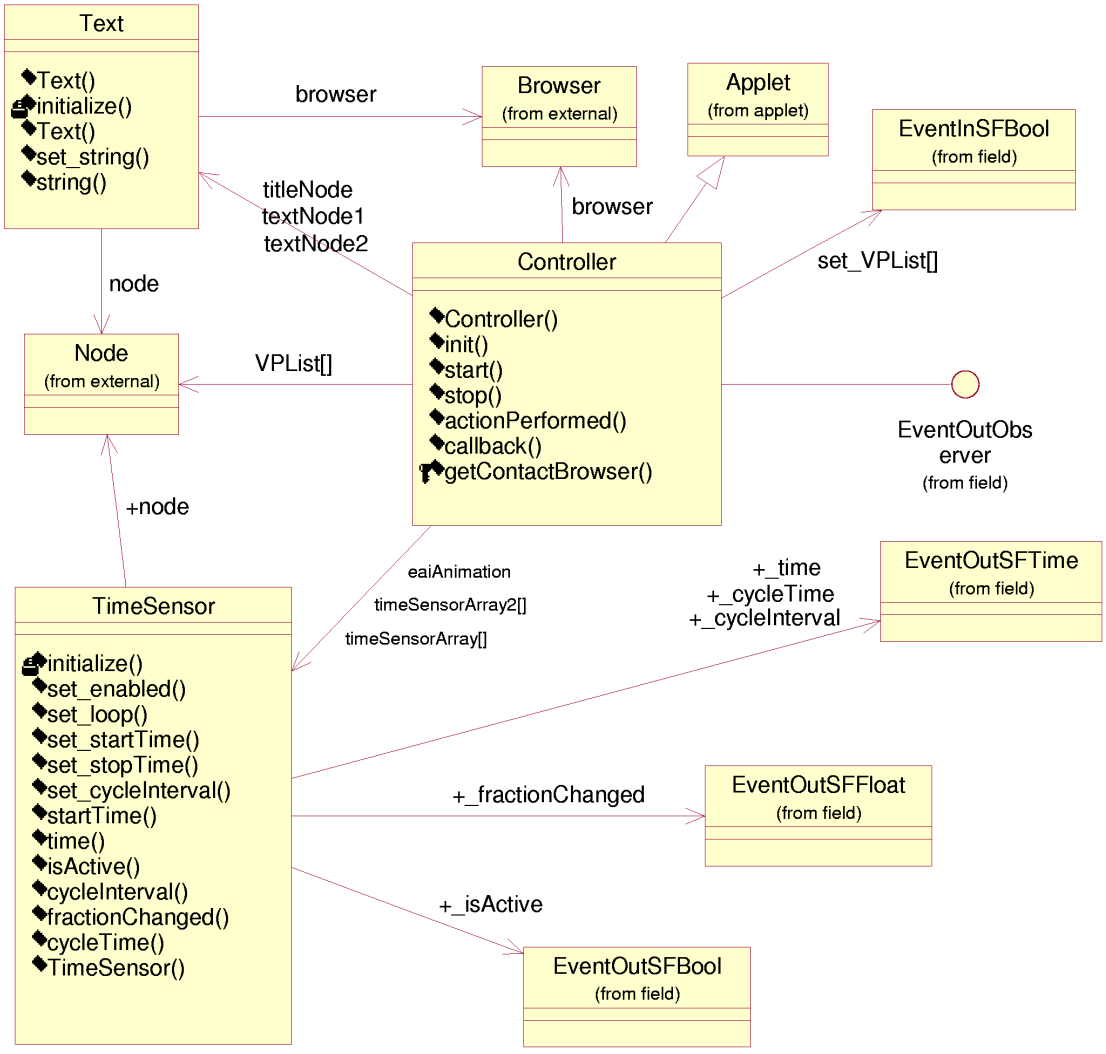

Abbildung 4.4.3: Klassendiagramm

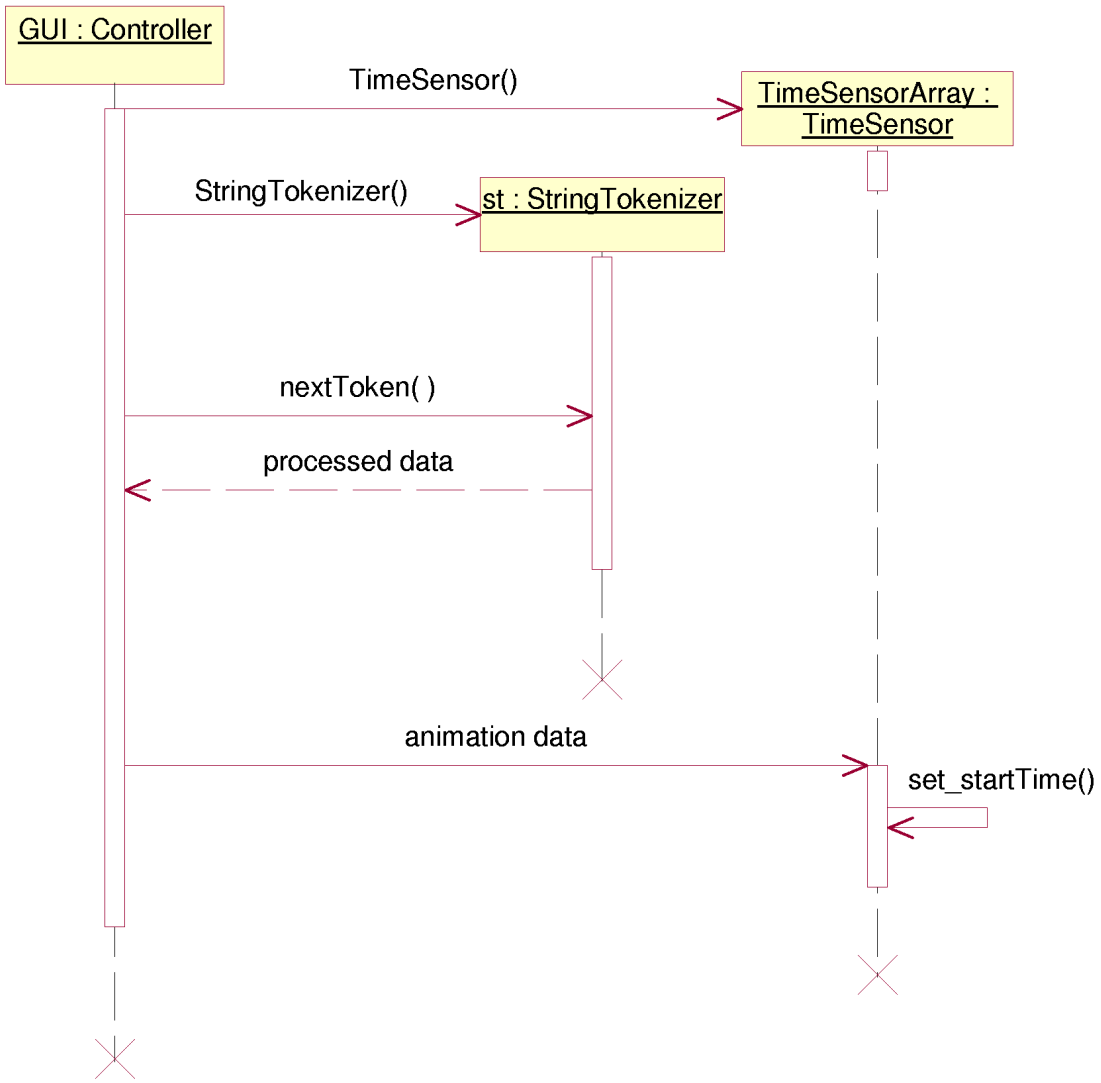

Abbildung 4.4.4: Sequenzdiagramm

5 Realisierung

Dieses Kapitel beschreibt die Implementation aller vorher diskutierten und analysierten Komponenten des VR-Systems.

5.1 Systemkonfiguration

Die Ausgangsituation war folgende: Auf dem zur Umsetzung des Prototyps genutzten Computer war ein Windows XP Professional mit Internet Explorer (IE) in der Version 6.0 - als Betriebssystem und Webrowser - installiert. Als VRML-Browser sollte der Cosmo Player von SGI zum Einsatz kommen, welcher jedoch nicht mit dem IE 6.0 zusammenarbeitete. Eine neuere, mit Windows XP/IE 6.0 kompatible, Version war zu Beginn der Arbeiten nicht verfügbar. Nachforschungen zufolge sollte Blaxxuns Contact in der Version 5.1 keine solcher Kompatibilitätsprobleme aufwerfen und nebenbei auch noch die am besten umgesetzte Implementation der VRML-Spezifikation besitzen. Nach einer Installation und einem kurzen Test, konnten keine Probleme bemerkt werden und Contact als VRML-Browser zur Entwicklung eingesetzt werden. Da weitere Softwareprobleme zu befürchten waren, wurde auf dem Computer zur Entwicklung Windows 2000 Professional SP1 als Betriebssystem installiert. Zur Modellierung der Szenenobjekte kamen Maya, CosmoWorlds - welches auch nicht auf der Windows XP Plattform lauffähig ist - und Spazz3D zum Einsatz.

Die Programmierung der Javaklassen wurde mit UltraEdit-32, einem Texteditor mit Syntax-Hervorhebung, durchgeführt.

5.2 Die VRML-Elemente

Da die Erstellung eines H-Anim konformen Avatars eine besondere Herausforderung darstellte, soll darauf nun im Folgenden genauer eingegangen werden. Die zusätzlichen Objekte ließen sich ohne schwerwiegende Probleme mit Maya[1] und CosmoWorlds realisieren, deshalb soll dies in diesem Kapitel nur kurz geschildert werden.

5.2.1 Erschaffung eines H-Anim Avatars

Die Erstellung des ersten Prototypen-Avatars wurde in folgenden Schritten realisiert:

- Erstellung der Skelettstrukturen in VRML
- Modellierung der Körperteile
- Einbindung der Körperteile in das Skelett
- Keyframen der Animationen
- Einbindung der Animation in die VRML-Szene
- Verbindung der Animationen mit „Triggern“ für die spätere Steuerung durch das Java-Applet
- Texturierung der Körperteile

Im Folgenden soll näher auf die aufgelisteten Arbeitsschritte eingegangen werden.

[1] vgl. Abschnitt „Aufgetretene Probleme“

5.2.1.1 Erstellung des Skeletts in VRML

Zu Beginn des praktischen Teils der Arbeit wurde versucht anhand der H-Anim-Spezifikation die Skelettstruktur zu erstellen. Dies erwies sich als schwieriger als vorab angenommen. Die Beispiele der Spezifikation sind wenig aufschlussreich und eindeutig, die Syntax und Verschachtelung der Gelenke in sich sind zu kompliziert, da beispielsweise bei einem LoA3-Avatar etwa 4500 Zeilen VRML-Quelltext generiert werden. Um die Avatare in dieser Arbeit zu realisieren, wurde auf die Fähigkeiten des Shareware-Modellierprogrammes Spazz3D zurückgegriffen, um somit die Arbeit an den Avataren zu erleichtern. Als Detaillevel wurde das LoA1 gewählt, welches 17 Gelenke beinhaltet, von denen aber nur 14 effektiv genutzt werden, wie Abbildung 5.2.1 illustriert.

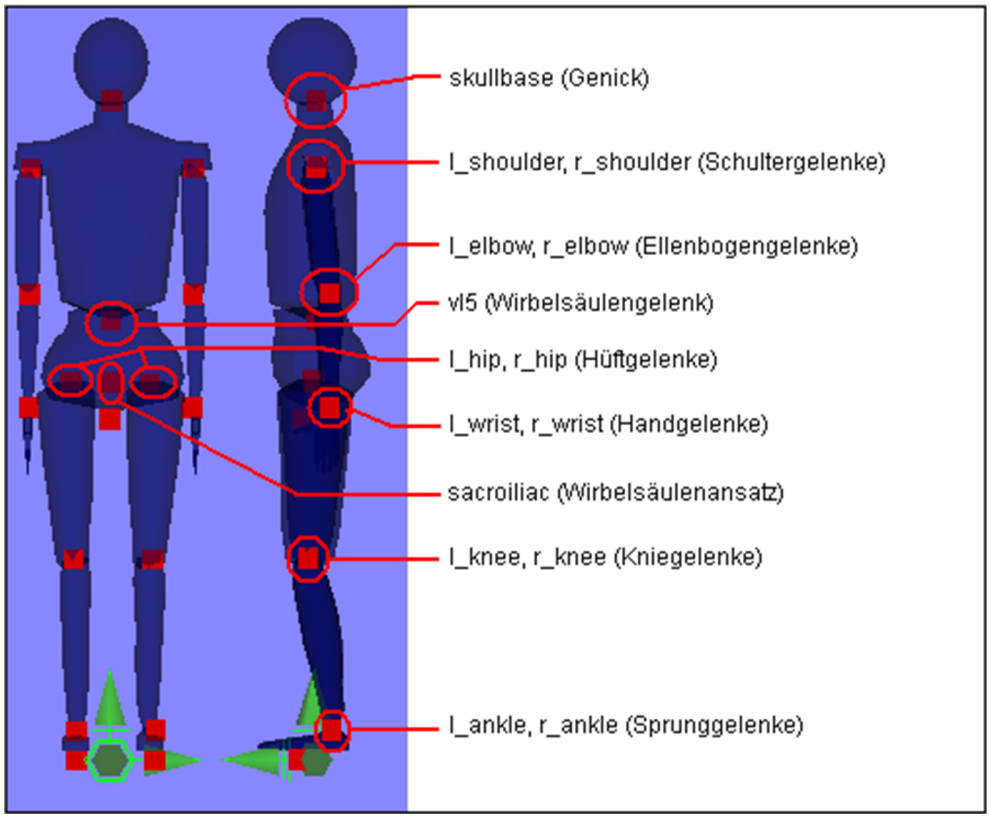

Abbildung 5.2.1: Gelenke des Avatars

In einem LoA1 Skelett sind prinzipiell alle Gelenke enthalten, um eine realistische Abbildung eines Humanoiden zu ermöglichen. Die weiterführenden, komplizierteren Detailstufen beinhalten jeweils nur noch feiner unterteilte Hände, Füße und Wirbelsäule, welche für das System jedoch nicht zwingend notwendig waren, bzw. die Performance des Gesamtsystems beeinträchtigt hätten.

5.2.1.2 Modellierung der Körperteile

Da die Modellierungsfähigkeiten von Spazz3D oder CosmoWorlds sich nicht wirklich mit einem professionellen 3D-Modeller messen können, kam zum Erstellen der Geometrien Maya 4 von Alias/Wavefront[2] zum Einsatz. Zu Beginn der Modellierung wurde versucht, einen möglichst realistischen menschlichen Körper zu erschaffen. Der angestrebte Realismus lässt sich mit einem sehr feinen Netz aus Polygonen annähern. Problematisch war jedoch, dass das Exportieren in das VRML-Format eine riesige Dateigröße bewirkte. Das Resultat war ein außerordentlich umfangreicher, 3000 Quelltext-Zeilen langer, `IndexedFaceSet`-Geometrieknoten. Dieser war zwar der Form nach analog zu dem Maya-Modell, jedoch geriet das Verhältnis zwischen Optik, Effizienz und Performance aus dem Gleichgewicht. Es wurde ein anderer Ansatz gewählt: Mit dem fein definierten Körper als Schablone wurden so mit einer speziellen Maya Technik, dem Face-Extrude Tool[3], aus einfachen Grundkörpern, wie z.B. Quadern, Körperteile geformt.
Mit Hilfe eines weiteren Mayatools, dem Polygon-Split Tool[4], konnten dann die an die Polygon-Hülle angenäherten Geometrien bei Bedarf weiter aufgeteilt und verfeinert werden.

[2] www.aliaswavefront.com
[3] Einzelne Flächen können mit dieser Technik aus der Geometrie gezogen werden.
[4] Mit dieser Technik können Flächen freihändig geteilt werden.

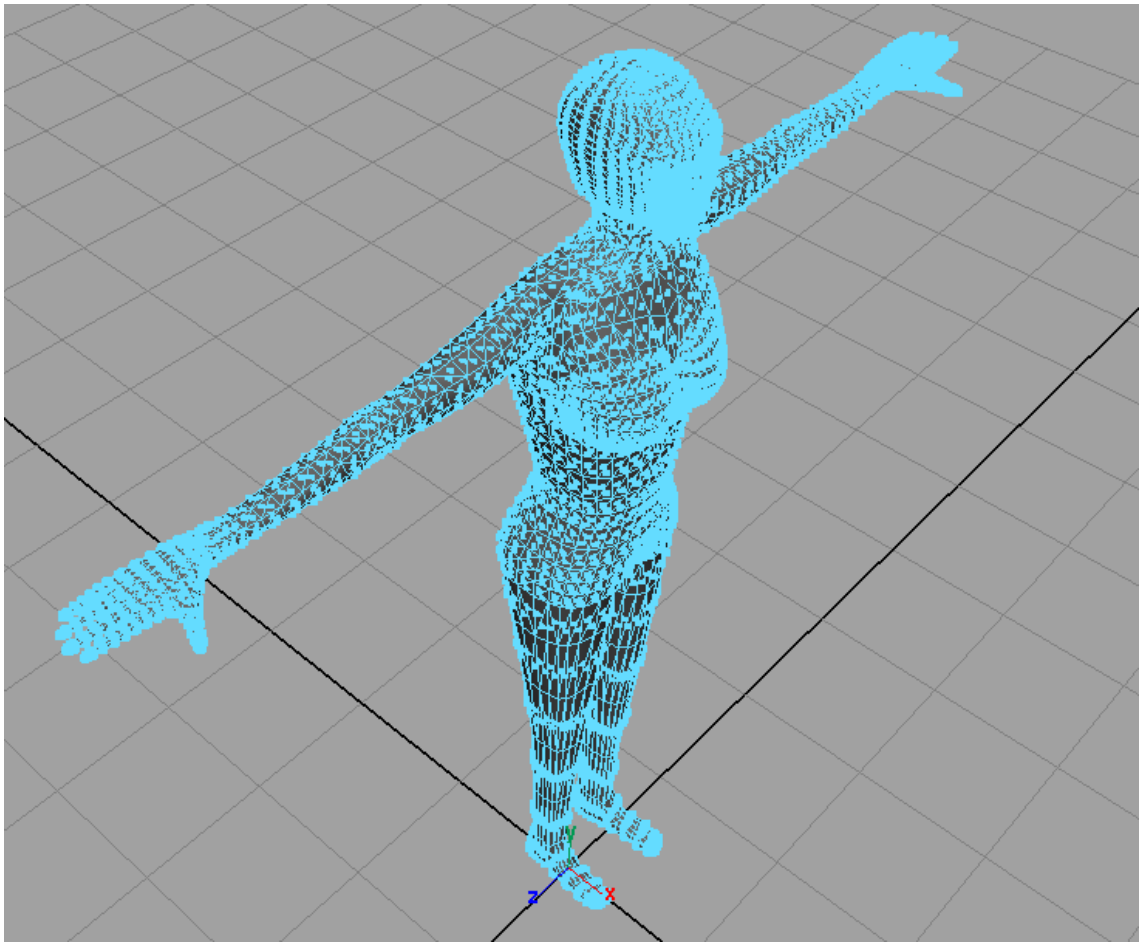

Abbildung 5.2.2: Polygon-Mesh des menschlichen Körpers

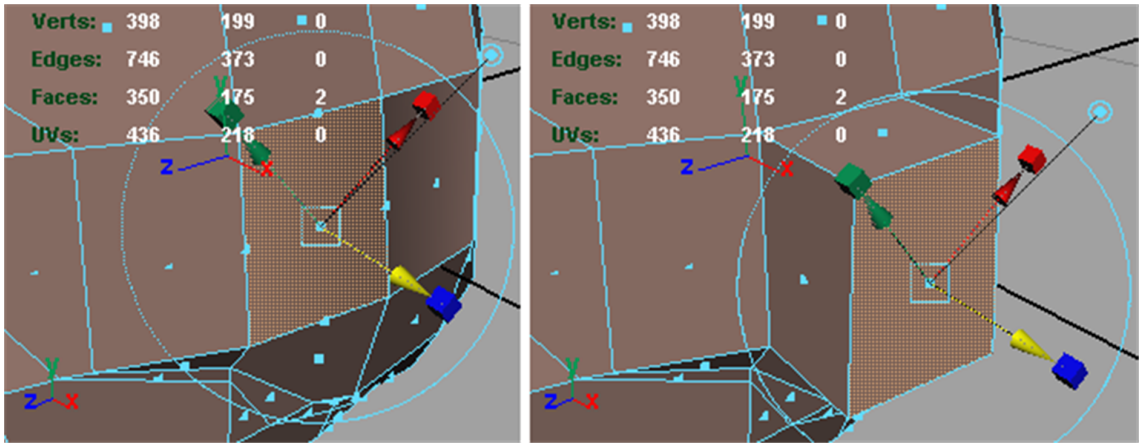

Abbildung 5.2.3: Polygon Extrude Tool von Maya

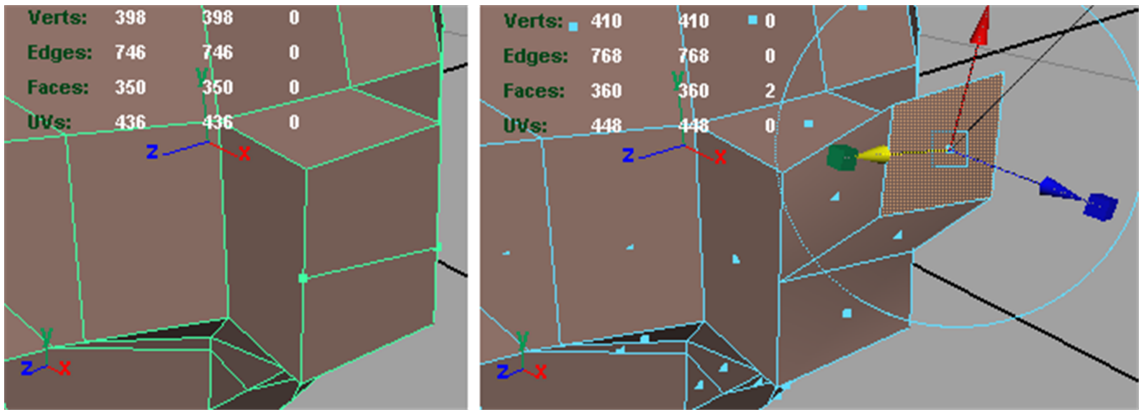

Abbildung 5.2.4: Polygon Split mit anschließendem Extrudieren

Als Resultat konnte somit die Anzahl der Polygone von 8000 auf unter 1000 reduziert werden, was in Performancesteigerungen bei der Darstellung sowie der Übertragung der Daten über eine Internetverbindung resultiert. Die Segmente wurden dann einzeln in das VRML-Format exportiert und bei Bedarf noch einmal in CosmoWorlds nachbearbeitet.

5.2.1.3 Einbindung der Körperteile in das Skelett

Sobald das Skelett syntaktisch korrekt vorlag, konnten Segmente an die jeweiligen Gelenke angebunden werden. Der Segmentknoten wird dazu als Children des Jointknotens angelegt, die Rotationspunkte (normalerweise die zentralen Endpunkte des Segments) in die jeweils über- und untergeordnete Gelenkposition verschoben und skaliert. Dadurch entstehen keine Zwischenräume zwischen den Segmenten und eine Rotation eines Körperteils wird nicht zu Zwischenräumen der Geometrien untereinander führen. Die Segmente wurden als `Inline`-Knoten importiert, um ein Modifizieren oder das komplette Austauschen zu ermöglichen. So bleibt die eigentliche Szene unberührt.

5.2.1.4 Keyframen der Animationen

Nachdem die Segmente an den richtigen Stellen platziert waren und der Avatar nun komplett vorlag, konnte mit dem Animieren begonnen werden. Hierzu kam Spazz3D zum Einsatz, das sich als hervorragendes VRML-Werkzeug erwiesen hat. Spazz3D verfügt über eine Funktion, die das Erstellen von speziellen H-Anim Animation ermöglicht. Dieses Animationswerkzeug stellt eine grafische Oberfläche zum manuellen Bearbeiten der Position und Rotation von Segmenten zur Verfügung. Dies kann sowohl per Maus, als auch durch numerische Eingaben von Rotationsachse und Winkel sowie von Translationswerten geschehen.

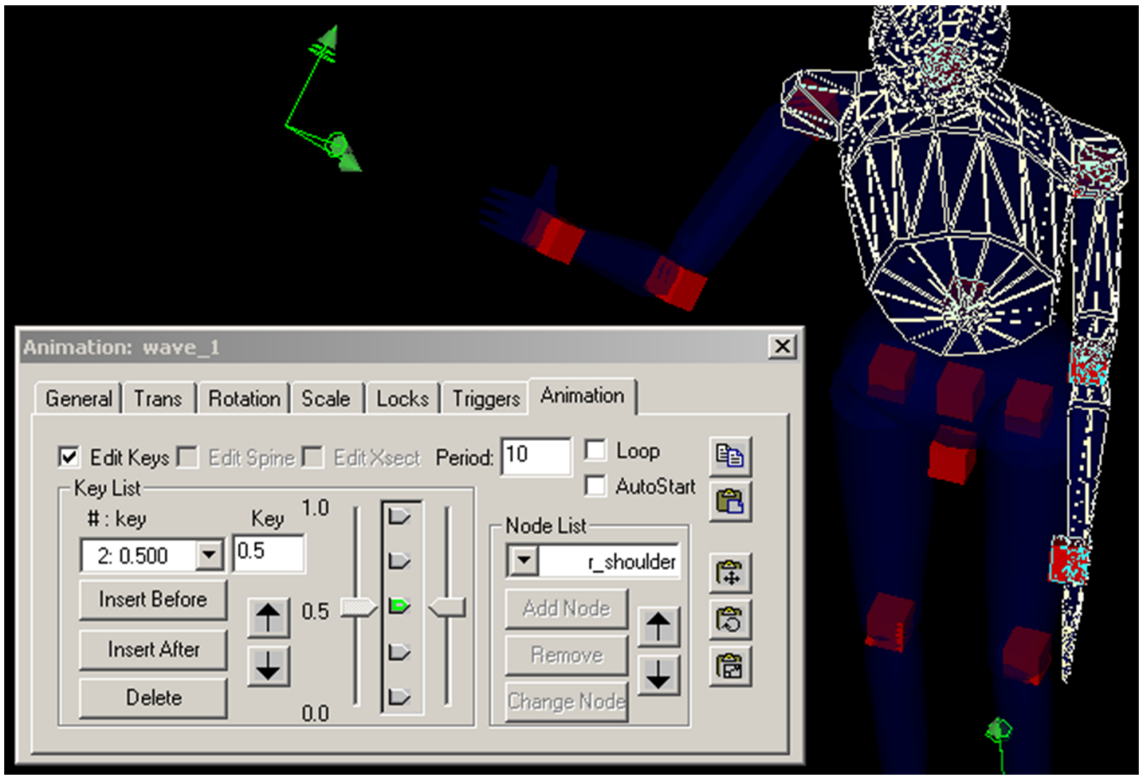

Abbildung 5.2.5: Editor zur Keyframe-Animation des Avatars

Wie in Abbildung 5.2.5 zu sehen ist, stellt das Animationswerkzeug einige weitere wichtige Funktionen zur Verfügung. So können Keyframes eingefügt und entfernt sowie die zugehörigen Zeitwerte modifiziert werden. Optionen wie die Wiederholung der Animation und ein automatisches Abspielen können ebenso aktiviert werden. Dabei wird der zugehörige TimeSensor um das `loop = TRUE`-Attribut erweitert. Die Möglichkeit Joints bzw. ganze Jointgruppen direkt zur Rotation und Translation auswählen zu können, ist wohl die wichtigste Funktionalität.

5.2.1.5 Verbindung der Animationen mit „Triggern“

Im Abschnitt 3.5.2.1 wurde bereits beschrieben, wie Animation und Interaktion in VRML97 umgesetzt sind. Ein Abspielen der Animationen durch die Steuerungselemente des Java-Applets wird durch den VRML-Knoten `TimeSensor` ermöglicht. So wurde ein `TimeSensor` an die jeweiligen Bewegungen der Avatare gebunden, die durch dessen Aktivierung stimuliert und abgespielt werden. Die mit dem `DEF`-Befehl festgelegte Bezeichnung des `TimeSensors` ermöglicht die Deklaration im gesamten Szenenbaum, der Wert `cycleInterval 2.0` definiert das Versenden von Zeitereignissen für die Dauer von zwei Sekunden.

```
DEF avatar1anim1 TimeSensor { cycleInterval 2.0 }
```

Für die Ausführung einer Rotation in dem herangezogenen Beispiel, werden zuerst alle Zeitereignisse des `TimeSensor`-Knotens in den `OrientationInterpolator` geroutet. Der Interpolator ist wiederum an den zu rotierenden Jointknoten des Avatar-Skeletts gebunden. Der folgende Code-Ausschnitt illustriert die Verbindung von `TimeSensor` zu `OrientationInterpolator` und von diesem zum H-Anim Joint `vl5` (Gelenk zwischen Wirbelsäule und Becken):

```
ROUTE avatar1anim1.fraction_changed TO vl5RotHi.set_fraction
ROUTE vl5RotHi.value_changed TO hanim_vl5.set_rotation
```

Der zugehörige `OrientationInterpolator` ist folgendermaßen implementiert:

```
DEF vl5RotHi OrientationInterpolator {
             key [ 0, 0.25, 0.75, 1 ]
             keyValue [ 1 0 0 0,
                        1 0 0 -0.2,
                        1 0 0 -0.2,
                        1 0 0 0
                      ]
             }
```

Der `OrientationInterpolator` im Codeausschnitt ist mit vier Keyframes und zugehörigen Rotationsinformationen (Rotationsachse und der Rotationswinkel in rad[5]) ausgestattet und bezieht sich lediglich auf die Rotation eines Gelenks. Durchschnittlich besteht eine Animation der Prototyp-Avatare aus 14 unabhängigen Translationen und Rotation. Das bedeutet, dass der TimeSensor, welcher die Animation auslöst, insgesamt 14 mal an die entsprechenden Interpolatoren geroutet werden muss. Diese 14 Interpolatoren sind wiederum an die zu bewegenden Jointknoten geroutet. Bei einer Anzahl von zehn definierten Animationssequenzen pro Avatar sind also durchschnittlich 280 `ROUTE`-Befehle im VRML-Code richtig zu verschalten.

5.2.2 Die Szene

Die VRML-Szene kann realistischer gestaltet werden, indem man noch weitere 3D-Objekte integriert. Zum Großteil wurden diese Objekte in Maya modelliert, teilweise dort auch schon mit einem Material bedacht und texturiert. Da Maya aber unnötigen und teilweise sogar fehlerhaften VRML-Code exportierte, wurden diese Objekte in CosmoWorlds und die Quelltexte in UltraEdit-32 nachbearbeitet.

[5] Radiant - die Maßeinheit des Bogenmaßes.

In Maya wurden zum Beispiel die Bühne, die Bäume, die Kirche[6] und die Pflanzenkästen modelliert. Für die kleinen Pflanzen kam ein neues Programm namens Virtual Flower[7] zum Einsatz, welches eine Fülle von Optionen zum Kreieren von Pflanzen in VRML bietet. In der Vollversion, welche für etwa 25 Euro verfügbar ist, können diese Pflanzen sogar noch umfangreich animiert werden, was erstaunliche Effekte ermöglicht.

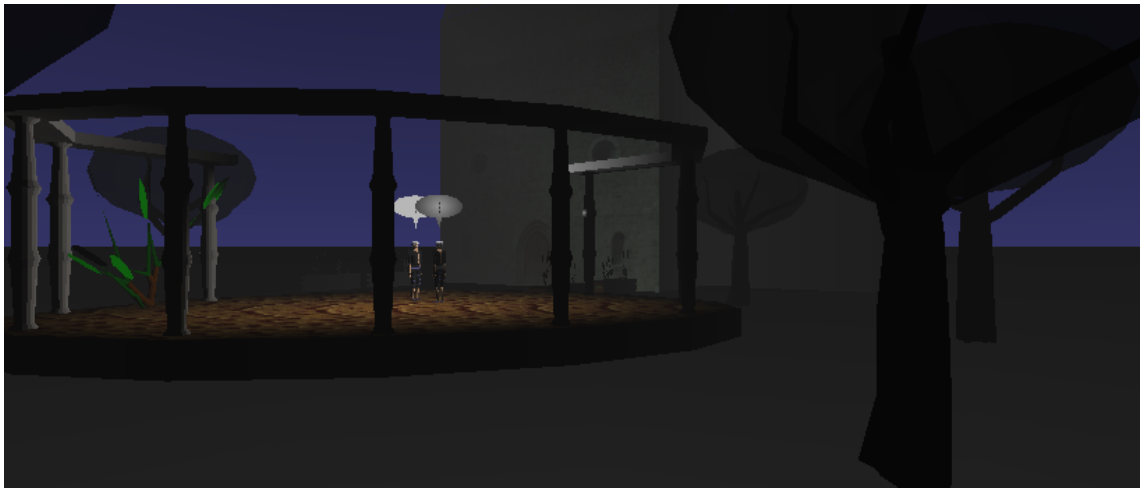

Abbildung 5.2.6: Die VRML-Szene mit Objekten

Ein Großteil der zusätzlichen 3D-Objekte sind per `Inline`-Knoten in die VRML-Szene integriert. Dies ermöglicht die Austauschbarkeit der Objekte, ohne die Hauptdatei selbst verändern zu müssen und belässt diese ebenfalls in einem überschaubaren Umfang (obwohl dieser trotzdem noch etwa 4000 Zeilen VRML-Code beträgt).

[6] Bei der Kirche handelt es sich um ein Modell der Nikolaikirche aus dem 18. Jahrhundert, welches vom Diplomanden im Zuge eines vorhergehenden Projektes in Zusammenarbeit mit zwei Kommilitonen modelliert wurde.

[7] Ein Download ist unter http://homepage.ntlworld.com/robertwalker/virtualflower möglich.

5.3 Webfrontend

Das Webfrontend ist der Bereich des Systems in dem der Benutzer sein Theaterstück aus Bausteinen zusammensetzen kann. Mit den eingebetteten Formularelementen lassen sich folgende Elemente definieren:

- Autor, Titel und Szene[8] des Stückes,
- die Bewegung, welche der jeweilige Avatar auszuführen hat,
- der Text beider Avatare,
- die Kamera, welche zum Zeitpunkt dieser Bewegung aktiv ist.

5.3.1 Aufbau und Funktionen

Das Frontend ist aus den HTML-Formularelementen Button, Pull-Down-Menü und Textfeld zusammengesetzt. Diese Formelemente ermöglichen es dem Benutzer, über hinterlegte JavaScript-Funktionen, die Regieanweisungen für die VRML-Szene zusammenzustellen.

5.3.1.1 Formularelemente

Die Pull-Down-Menüs sind jeweils mit Optionen und zugehörigen Werten gefüllt. Die eindeutige Bezeichnung als `avatar1`,`avatar2` und `camera` ist notwendig, um bei dem Hinzufügen eines Skriptblocks die jeweilige Auswahl den richtigen Variablen zuzuordnen.

```
[..]
<select name="avatar1" class="select">
<option value="anim1" selected>Idle</option>
<option value="anim2">Verbeugen</option>
<option value="anim3">Kniefall ab</option>
<option value="anim4">Kniefall auf</option>
[..]
```

[8] oder anderweitige Informationen

5.3.1.2 JavaScript-Funktionen

Sobald der Button zum Hinzufügen eines Skriptblocks betätigt wird, werden per JavaScript-Funktion die ausgewählten Teile jeweils an die vorherigen Auswahlen konkateniert. Dazu werden die Blöcke an eine jeweils eigene Stringvariable geschrieben, welche gleichzeitig die Textfelder zur Statusausgabe innerhalb des Webfrontends repräsentieren.

```
function add() {
form.playA1.value = form.playA1.value
                    + "avatar1:" + form.avatar1.value
                    + "|avatar1text:" + form.text11.value
                    + " ;" + form.text21.value + " ;"
                    + form.text31.value + "|"
[..]
form.cam.value = form.cam.value + form.camera.value
}
[..]
<input type="button" class="b2" value="Hinzufügen" onClick="add()">
[..]
```

5.3.2 Kommunikation zum Applet

Bei Ausführung des Formulars werden die benötigten Variablen per `post`-Methode an ein PHP-Skript weitergegeben.

```
<form name="form" method="post" onSubmit="addTitle();" action=
 "Controller.php?playA1=$playA1&playA2=$playA2&cam=$cam&title=$title">
```

Auf diesem Wege werden die Variablen, welche Anweisungsblöcke enthalten, dem PHP-Dokument bekannt gemacht. Regieanweisungen und Titelinformationen werden dann als Parameter in das `APPLET`-Tag der HTML-Umgebung geschrieben.

```
<APPLET CODE="Controller.class" WIDTH="550" HEIGHT="170"
ALIGN="middle" NAME="Controller" MAYSCRIPT>
<?php
echo ("<PARAM NAME = \"playA1\" VALUE = \"" . $playA1 . "\">");
echo ("<PARAM NAME = \"playA2\" VALUE = \"" . $playA2 . "\">");
echo ("<PARAM NAME = \"cam\" VALUE = \"" . $cam . "\">");
echo ("<PARAM NAME = \"title\" VALUE = \" " . $title . "\">");?>
</APPLET>
```

Im Applet werden diese Parameter dann gezielt abgefragt, zerstückelt und an die jeweiligen Objekte geparst. Dazu jedoch in Abschnitt 5.4.2 mehr.

5.4 Applet zur Steuerung

Wie festgestellt werden konnte, ist Java zur Steuerung der komplexen Abläufe am besten geeignet. Um den Prototypen online verfügbar machen zu können, musste die Steuerungslogik in einem Java-Applet implementiert werden. Benutzerinteraktion wurde ermöglicht, indem das Applet mit einer grafischen Benutzerschnittstelle (Userinterface) ausgestattet wurde.

5.4.1 Appletfunktionen

Außer der Benutzerschnittstelle implementiert das Applet noch die eigentliche Steuerungslogik. Diese ist in Objekten, die per `Callback`-Funktion die Ereignisse in der VRML-Szene steuern, organisiert. Die vom Applet angenommenen Anweisungen werden innerhalb der `Start()`-Methode des Applets aufgeteilt und an die jeweiligen Objekte verteilt.

5.4.2 Benutzen der Befehlsobjekte

Wie in Abschnitt 5.3.2 erwähnt, werden die Anweisungsstrings per `PARAM`-Tag in die `embed`-Umgebung des Applets geschrieben, vom selbigen aufgegriffen und, falls sie nicht leer sind, dann jeweils in einem temporären String zwischengespeichert. Per Java-Funktion `StringTokenizer()` wird dieser in Tokens (Einzelteile) zerlegt und sukzessive in ein String-Array kopiert, wie folgender Codeausschnitt illustriert:

```
data = getParameter("playA1");
if (data != null) {
try {
          st = new StringTokenizer(data, ":|");
          playA1 = new String[st.countTokens()][4];
      for (int i=0; st.hasMoreTokens(); i++)
        {
        for (int j=0; j<4; j++)
        {
        playA1[i][j] = st.nextToken();
        }
          commandCount++;
            }
         } catch (NumberFormatException e) { }
     }
```

Da im Prototyp zwei Avatare implementiert sind, wurde für jeden Avatar im Voraus ein eigener Befehlsblock-String übergeben, so dass der zweite nach der gleichen Methode zu verarbeiten ist und in einem eigenen Stringarray gespeichert wird. Die Länge dieser Befehlstringarrays wird zur Laufzeit dynamisch allokiert, da die Anzahl der Befehlsblöcke variabel sein kann. Die vertikale Dimension ist von dem Aufbau der Blöcke bekannt: zwei Teilblöcke, jeweils mit Objektbezeichnung und deren zugehörigen Werten, was in einer Dimensionierung des Arrays von `playA1[Anzahl der Befehlsblöcke][4]` resultiert. Der Inhalt dieses Arrays bleibt während der Laufzeit der Applikation erhalten. Die globale Laufvariable `commandCount` dient zur Bestimmung der Dimensionen der Objekte, die bei Beginn eines Abspielens initialisiert werden müssen. Jedoch wird `commandCount` nur bei der Verarbeitung des ersten Parameterstrings vom Wert 0 inkrementiert, da jedes der Arrays die gleiche Länge besitzt.

```
set_ViewPointArray = new EventInSFBool[commandCount];
timeSensorArray = new TimeSensor[commandCount];
```

Im Codebeispiel wird anhand von `commandCount` die Länge der Arrays für Viewpoints und Animationen allokiert.

5.4.2.1 Spielen der Animationen

Im Abschnitt Design wurde bereits die Anatomie des Animationsobjektes (aus Klasse TimeSensor) herausgearbeitet. Bei einem Objektaufruf werden die `EventOut`-Felder des spezifizierten `TimeSensor`-Knotens aus der VRML-Szene (siehe Codebeispiel) extrahiert und Methoden zu deren Manipulation bereitgestellt.

```
// Felder des TimeSensor-Nodes
_isActive         = (EventOutSFBool)
                          node.getEventOut("isActive");
_cycleInterval    = (EventOutSFTime)
                          node.getEventOut("cycleInterval_changed");
_fractionChanged  = (EventOutSFFloat)
                          node.getEventOut("fraction_changed");
_cycleTime        = (EventOutSFTime)
                          node.getEventOut("cycleTime");
_time             = (EventOutSFTime)
                          node.getEventOut("time");
```

Nachdem das Objektarray für die zu spielenden Animationen initialisiert wurde, kann es mit den korrelierenden Daten gefüllt werden. Dazu wird, indiziert anhand von `commandCount`, ein Konkatenat[9] des aktuellen Befehlblockteils zum Initialisieren eines neuen `TimeSensor`-Objektes genutzt, welches anschließend in ein Array - bestehend aus `TimeSensor`-Objekten - geschrieben wird. Der Zugang zu den `TimeSensor`-Knoten ist durch die Definition der Bezeichner problemlos möglich. So wird im Webfrontend eine Animation aus `avatarX` und `animX` zusammengestellt, die jeweils durch einen TimeSensor mit der Bezeichnung `avatarXanimX` gestartet wird.

```
for (int i=0; i<commandCount; i++) {
 String temp = playA1[i][0] + playA1[i][1];
 timeSensorArray[tSensorSelection++] = new TimeSensor(browser,temp);
}
```

Das Objekt `timeSensorArray[]` beinhaltet nun alle vom Benutzer ausgewählten Bewegungsanimationen, die bei einem Abspielen der Szene

[9] Eine lineare Verkettung von verschiedenen einzelnen Strings.

ausgeführt werden. Innerhalb der `callback()`-Funktion werden dann die `TimeSensor`-Knoten der jeweiligen Avatare getriggert[10]:

```
timeSensorArray[tSensorSelection].set_startTime(zeit);
```

Der Aufbau des `TimeSensor`-Objektes ist zu Beginn dieses Abschnittes dargestellt worden. Es wurde auch erwähnt, dass die `TimeSensor`-Klasse Methoden zur Manipulation der Sensoren in der VRML-Szene bereitstellt. Mit dem im Codebeispiel gezeigten Methodenaufruf `objekt.set_startTime(zeit)` wird das Eingangsereignis (`EventIn`) des Sensors folgendermaßen stimuliert.

```
public void set_startTime(double value)
  throws InvalidEventInException
  {
   ((EventInSFTime) node.getEventIn("set_startTime")).setValue(value);
  }
```

Der `TimeSensor`-Knoten wird aktiviert und schickt in Intervallen Zeitereignisse an die jeweiligen Interpolatoren an die er geroutet ist. Die Animation wird abgespielt.

Sollte die letzte Animation abgespielt worden sein, wird die Laufvariable des `TimeSensorArray[]`-Objekts wieder auf den Wert 0 gesetzt, um so das Abspielen von Beginn an zu wiederholen.

```
if (tSensorSelection >= timeSensorArray.length) {
      tSensorSelection = 0;
      }
```

5.4.2.2 Änderung der Viewpoints

Die Änderung der Viewpoints ist ähnlich dem Abspielen der Bewegungsanimationen realisiert. Jedoch bestand aufgrund des relativ geringen Umfanges der Funktionen bzw. der Objekte kein Bedarf, diese Funktionalität in einer extra

[10] engl: stimuliert, aktiviert

Klasse auszulagern. Es wurde auch dagegen entschieden, dem Benutzer einen weitergehenden Eingriff auf die Kameraeinstellungen zu gewähren. Unter dem Gesichtspunkt, dass es einem Benutzer nicht zugemutet werden kann, sich Vorwissen aneignen zu müssen, um eine Kamera im Raum so zu platzieren[11] wie er es wünscht, werden möglichst viele Viewpoints vordefiniert. Auch ist eine interaktive Platzierung von Kameras per Hand (vergleichbar zum Modellierungsprozess einer Szene) innerhalb einer VRML-Szene, soweit bekannt, nicht möglich.

Um die, für die Szene relevanten Kameraeinstellungen möglichst einfach aus der VRML-Szene extrahieren zu können, wurde auf einen kleinen Trick zurückgegriffen. Alle Viewpoints die Kamerapositionen repräsentieren, sind in einer Gruppe `Viewpoints` organisiert. Per `DEF`-Befehl ist diese Gruppe dann im gesamten VRML-Dokument deklariert. Der Ansatz, die gesamte Anzahl der Viewpoints und ihre Node-Referenz, die bisher der Steuerungslogik unbekannt sind, nun in ein Feld von Kamerapositionen, konform zu der, durch den Benutzer getroffenen Auswahl abzubilden, ist ähnlich dem der Bewegungsanimationen.

```
Node[] VPList;
[..]
//Viewpoint-Gruppe referenzieren
Node Viewpoints = browser.getNode("Viewpoints");
VPList = ((EventOutMFNode)
            (Viewpoints.getEventOut("children_changed"))).getValue();
```

Dem Quelltext ist zu entnehmen, dass ein Array vom Datentyp MFNode (Multi Field Node) mit Referenzen auf die einzelnen Viewpoint-Knoten der Group „Viewpoints“ gefüllt wird. Das referenzierte `EventOut` ist das `children_changed`-Ereignis. Im Folgenden Codeausschnitt wird die Liste der vom Benutzer ausgewählten Viewpoints gefüllt:

[11] Eine Kamera im Raum besitzt neben einer Positionskoordinate, einem Richtungsvektor zu der Blickrichtung und einem Winkel zur Weite des Objektives/Blickfeldes, auch Attribute wie isActive.

```
for (int j=0; j < commandCount ; j++) {
   set_VPList[j] = (EventInSFBool)
         VPList[Integer.parseInt(cam[j][1])].getEventIn("set_bind");
  }
```

Aus dem Anweisungsstring `cam[][]` werden die Kamerapositionen, welche der Benutzer vorher im Frontend ausgewählt hat, innerhalb der Liste aller Viewpoints als Index zur Auswahl benutzt. Hier trat erstmalig ein Problem mit den Datentypen auf. Der Anweisungsstring ist vom Datentyp String, während Indizes vom Datentyp Integer sein müssen. Letztendlich handelt es sich bei der Kameraposition nur um eine Integerzahl, die jedoch als String repräsentiert wird. Abhilfe in dieser Situation schuf die Möglichkeit Stringdaten per Datentyp-Cast in eine Integerzahl umzuwandeln. Die Javafunktion `Integer.parseInt(String)` ermöglicht das. Sobald die Indizierung der Viewpoint-Liste durch das Umwandeln funktionierte, konnten die `set_bind`-Ereignisse, welche für den Zustand[12] eines Viewpoints verantwortlich sind, per `EventInSFBool`-Typecast in die sortierte Liste der anzuzeigenden Kamerapositionen geschrieben werden. Die Werte dieser Liste sind vom Datentyp Boolean, welche in der `Callback()`-Funktion des Applets jeweils auf `TRUE` gesetzt werden.

```
EventInSFBool set_bindVP = set_VPList[view];
set_bindVP.setValue(true);
```

Mit Setzen der Variable `set_bind`VP auf `TRUE` wird der neue, aktive Viewpoint festgelegt und dadurch die Kameraposition, falls sie von der vorherigen abweichen sollte, verändert. In der Knotendefinition der Viewpoints innerhalb des VRML-Dokuments ist die Option „jump“ deaktiviert, so dass die Kamera des Betrachters fließend in die neue Position verschoben - und gegebenenfalls gedreht - wird. Wäre diese Option aktiviert, würde die Kameraperspektive einfach nur abrupt verändert werden. Da dies aber auch ein gewünschter dramaturgischer Effekt sein könnte, wäre es durchaus denkbar diesen Parameter auch

[12] set_bind = TRUE für aktiv und FALSE für inaktiv

veränderbar zu machen und somit dem Benutzer eine größere Handlungsfreiheit einzuräumen. Davon wurde vorerst jedoch aus Gründen der vereinfachten Bedienung des Webfrontends Abstand genommen, bis vielleicht eine ergonomisch günstigere Möglichkeit entwickelt werden kann, um eine Vielzahl der parametrisierbaren Optionen überschaulicher zu machen.

5.4.2.3 Änderung des Textes

Die Funktion, den angezeigten Text der Avatare zu ändern, ist aufgrund der Komplexität wieder in einer externen Klasse implementiert. Die Klasse an sich realisiert drei Hauptfunktionen:

- Ein Objekt mit einer Referenz auf einen VRML-Textknoten zu initialisieren.
- Eine Funktion zum extrahieren von Text aus der Szene und
- eine Funktion zum Ändern des Textes im spezifizierten Textknoten.

In dem folgenden Codebeispiel ist zu sehen, wie ein Textobjekt initialisiert wird. Dieses Objekt referenziert den VRML-Textknoten `AVATARTEXT1`, welcher auf der Sprechblase des ersten Avatars platziert ist.

```
Text textNode1;
[....]
textNode1 = new Text(browser,"AVATARTEXT1");
```

Den Ausführungen zur Modellierung konnte entnommen werden, das drei Zeilen von Text für die Textknoten vorgesehen sind. Da die Struktur der Befehlsblöcke diese drei Zeilen jedoch nur in einem String zulassen, musste auf einen Umweg zurückgegriffen werden. Im Frontend sind so drei Textfelder für die Texteingaben vorgesehen, welche beim Zusammenstellen eines Anweisungsblockes wiederum zu einem String konkateniert, jedoch innerhalb dieses Konkatenats durch `Limiter` in der Form von Semikola logisch getrennt sind. Im Applet wird dieser String dann mit einem weiteren `StringTokenizer()` aufgeteilt und die so

entstandenen Teile in ein Stringarray geschrieben, wie dem folgenden Code zu entnehmen ist.

```
sd = new StringTokenizer(playA1[tSensorSelection][3], ";");
a1speech = new String[3];
for (int i=0; sd.hasMoreTokens(); i++)
 {
     a1speech[i] = sd.nextToken();
 }
```

Sollten keine oder nicht alle der drei Zeilen des Textes zur Wiedergabe genutzt worden sein, wird ein Leerzeichen in die leeren Arrayteile geschrieben, um eine Exception[13] zu verhindern. Dem nächsten Codeausschnitt zufolge wird der Inhalt des Bufferstrings in das Stringarray `strArrayToNode` geschrieben, welches dann per `set_string()`-Methode in den VRML-Textknoten kopiert wird.

```
String[] strArrayToNode = {a1speech[0],a1speech[1],a1speech[2]};
textNode1.set_string(strArrayToNode);
```

Das `set_string`-EventIn ist als `EventInMFString`-Datentyp deklariert. MFString ist der VRML-Feldtyp[14], in den der Inhalt des Textarrays abgebildet wird.

```
public void set_string(String[] value)
   throws InvalidEventInException {
 ((EventInMFString) node.getEventIn("set_string")).setValue(value);
}
```

Da in der VRML-Szene derzeit zwei Avatare und deren Sprechblasen platziert sind, werden die zuvor genannten Mechanismen - wie bei den Bewegungsanimationen auch - zweimal ausgeführt.

5.5 Aufgetretene Probleme

Zu Beginn der 3D-Modellierungsphase wurde der Autor gleich mit dem ersten Problem konfrontiert: Der VRML97-Exporter von Maya 4.0 arbeitete nicht

[13] Engl: Ausnahme, Ausnahmeregelung; Laufzeitfehler eines Java-Programms, bspw. durch falsche Funktionsaufrufe oder Datentypenbenutzung.

[14] Datentyp

fehlerfrei. So wurden einige der exportierten Objekte in falschen, unspezifizierten und unbekannten Knotentypen definiert. Das hatte zur Folge, dass bei dem Laden in einem VRML-Browser oder zur Nachbearbeitung in CosmoWorlds Syntaxfehler eine Darstellung verhinderten. Nach einem Laden der Datei in UltraEdit-32 wurden dann per Hand die entsprechend fehlerhaften Codeteile berichtigt.

Eine weitere Eigenheit des Maya 4.0 Exporters war es, dem Szenenbaum unnötige Kopien der modellierten Geometrien anzuhängen. Sobald etwaige Syntaxfehler behoben waren, konnte eine solche Szene in CosmoWorlds geladen werden und die überflüssigen Objekte einfach aus der Objektliste gelöscht werden. Dadurch konnten jeweils bis zu 70% VRML-Code eingespart werden, was angesichts der Fülle der Daten schon eine beträchtliche Optimierung darstellte.

Ein weiteres Problem, dass bei der Modellierung und dem Export in bzw. aus Maya auftrat, war die Tatsache, dass duplizierte Objekte innerhalb von Maya grundsätzlich als Instanzen der Vorlage geschaffen werden. Dies hatte zur Folge, dass bei dem Export zusammengesetzer Objekte Geometrien fehlten. Nachdem die Dupliziermethode auf „duplicate copy world“ umgestellt wurde, war das Exportresultat jedoch zufriedenstellend und alle modellierten Objekte in der VRML-Szene abbildbar.

Ein Problem, das bei der Benutzung von Text in VRML auftrat, betraf die Ausdehnung der Zeichenketten innerhalb der Textknoten. Es ist nicht Bestandteil der Textknotenspezifikation, dass die maximale Anzahl von Zeichen in einem SFString- oder MFString-Feld festgelegt werden kann. In dem vorliegenden VR-System wäre es jedoch wünschenswert gewesen, diese Möglichkeit zur Verfügung gehabt zu haben und demnach Stringelemente, die die maximale Länge überschreiten jeweils abzuschneiden und den Nachfolgenden voran zu stellen. Stattdessen werden in VRML bei zunehmender Anzahl von Zeichen

innerhalb eines Textfeldes die Buchstaben zusammengestaucht. Da dieser Effekt nicht gewollt ist, wurde die maximale Anzahl von Zeichen schon in den Formularelementen des Webfrontends limitiert. Diese Lösung ist zwar nicht professionell, jedoch lassen sich damit die Unzulänglichkeiten der VRML-Spezifikation umgehen.

Ein weiteres Problem trat im Zusammenhang mit dem Java-Programm auf: Das erste Kompilieren der Java-Steuerung schlug fehl, da die zur Entwicklung benötigten Java-Klassen für das VRML-EAI nicht gefunden werden konnten. Blaxxun zufolge sollten diese jedoch bei der Browserinstallation in die notwendigen Verzeichnisse kopiert werden. Ein Durchsuchen der Javaverzeichnisse offenbarte, dass alle vom EAI benötigten Klassen und Interfaces ordnungsgemäß installiert waren. Bei einem genaueren Betrachten des Klassenpakets stellte sich jedoch heraus, dass selbst die neueste Version von Contact mit den veralteten und verworfenen Package-Bezeichnungen arbeitete. Dies stellte die Programmierarbeit vor ein Problem: Durch die veraltete Bezeichnung des Packages von `vrml.external.*` konnte das Javaprogramm nicht spezifikationskonform mit Imports der Form `vrml.eai.*` kompiliert werden. Zu Hilfe kam jetzt ein anderer VRML-Browser, der die EAI-Spezifikation genauer umsetzt: Cortona von Parallel Graphics. Das EAI-Package von Cortona stellte sich als konform zu der EAI-Spezifikation [vgl. Web3D B] heraus, und konnte durch das Paket von Contact ersetzt werden. Das Kompilieren der Java-Quelltexte gelang nun problemlos.

6 Ergebnisse

6.1 Systemtest

Da im Laufe der Implementierungsphase schon Teile des VR-Systems funktionsfähig waren, wurden parallel zu der praktischen Umsetzung permanent Tests durchgeführt. Einerseits, um die Änderungen und Fortschritte zu visualisieren und andererseits zur Abstimmung der grafischen Inhalte. Da einige der verwendeten Technologien zur Realisierung des Prototypen dem Autor unbekannt waren, konnten die einzelnen Systemkomponenten so im Zuge der Umsetzung iterativ weiterentwickelt und verbessert werden.

Sobald das System konform zu der Analyse und den Designüberlegungen fertig gestellt war, konnte es auf verschiedenen Computersystemen auf Kompatibilität und Performance untersucht werden. Dazu wurden Computersysteme verschiedener Leistungsklassen mit Blaxxun Contact als VRML-Plug-In ausgestattet, und ein vollständiger Systemtest des online verfügbaren Prototypen durchgeführt. In Tabelle 6.1.1 sind die benutzten Testsysteme und ihre Hard- und Softwarekonfiguration, sowie deren objektive und subjektive Performance aufgeführt.

Systemeigenschaften	System 1	System 2	System 3
CPU (Intel)	P3 933MHz	Mobile P3 1GHz	Mobile P4 1.6GHz
RAM	512MB (SD)	128MB (SD)	256MB (DDR)
Grafikchip	nVidia GeForce 4 Ti4200	ATI Rage Mobility-P AGP	nVidia GeForce 4 420Go 32M
Betriebssystem	Windows XP SP1	Windows 2000 SP2	Windows 2000 Pro SP1
Webbrowser	IE 6.0	IE 5.0	IE 5.5
Lauffähigkeit	ohne Probleme	nach Änderung einiger Systemeinstellungen* ohne Probleme	ohne Probleme
CPU-Auslastung	50-70%	75-90%	40-60%
subjektive Darstellung	flüssig	flüssig	flüssig
* Auf diesem System war Java 1.4 installiert, welches die Microsoft JVM des Internet Explorers ausschaltet. Nach der Deaktivierung des Java Runtime Environments (JRE) wurde zum Abspielen des Applets die Microsoft JVM benutzt.			

Tabelle 6.1.1: Computersysteme zum Test

Die Tests wurden folgendermaßen durchgeführt: Nach der Installation von Blaxxun Contact wurde der Computer neu gestartet um evtl. performance-beeinflussende Faktoren, wie die Hauptspeicherallokierung durch unabhängig vom Test gelaufene Programme, ausschließen zu können, und so die uneingeschränkte Funktionstüchtigkeit des VRML-Plug-Ins zu gewährleisten. Falls zu diesem Zeitpunkt noch keine Internetverbindung bestand, wurde diese aufgebaut. Sobald das Webfrontend geladen war, wurden die gewünschten Steuerungsparameter eingegeben bzw. eingestellt und der Aufbau der VRML-Szene (vgl. Abbildung 6.1.1) und des Applets gestartet.

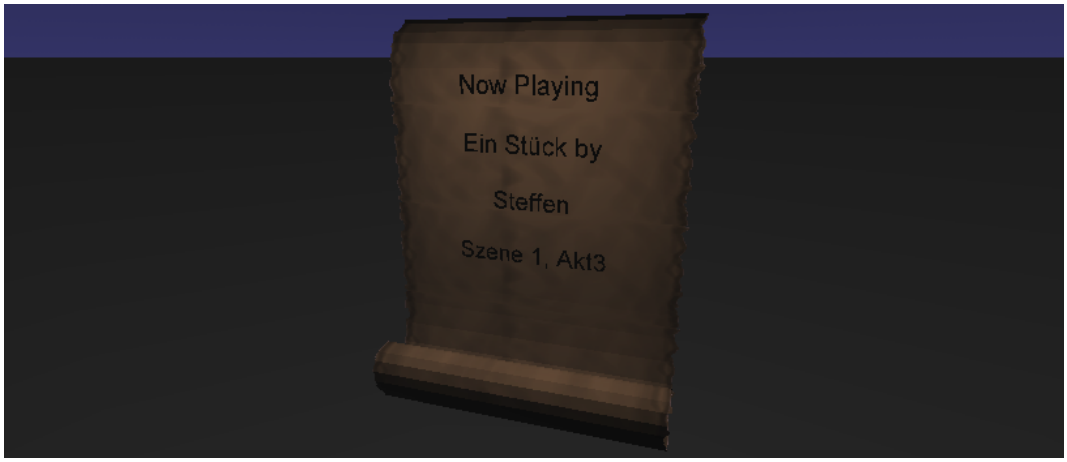

Abbildung 6.1.1: Eingangszustand des VRML-Theaterstücks

Sobald der „Play-Button“ des Applets betätigt wurde, begann das Abspielen des Stücks. Die folgende Abbildung illustriert eine Momentaufnahme aus der Szene:

Abbildung 6.1.2: Das abgespielte Stück

6.2 Fazit und Ausblick

Das Ziel dieser Arbeit war es, die dynamische Steuerung von Avataren am Bespiel eines Theaterstücks zu untersuchen. Mit dem entwickelten Prototypen ist es möglich, eine begrenzte Anzahl von Animationen nach belieben zu arrangieren und somit ein bereits definiertes Theaterstück darzustellen, oder aber eine eigenes Stück zu erschaffen. Leider war es nicht möglich, die Avatare an beliebige Positionen in der VRML-Szene „laufen“ zu lassen. Die Problem ist analog zu dem der freien Viewpoint-Einstellungen: Es kann nicht vorausgesetzt werden, dass ein Benutzer - ohne fundierte Kenntnisse zu 3D-Computergrafik und VRML - interaktiv Rotationswinkel und Translationsvektoren für eine derartige Manipulation festlegen kann. Jedoch is es aufgrund der Modularität des Systems mit einem VRML-Programm wie CosmoWorlds möglich, eigene Animationen und Kamerapositionen einzubinden und das Aussehen der Akteure zu verändern. Der Prototyp kann auf der Grundlage der VRML97-Spezifikation beliebig erweitert werden, um so auch anderen Anforderungen gerecht zu werden. Denkbar ist die Nutzung für interaktive 3D-E-Learning-Systeme, umfangreiche Führungen durch virtuelle Gebäude oder auch dynamisierte 3D-Onlineshop-Lösungen, die mit Avataren als Verkäufer realisiert sind.

Es wäre auch von Vorteil, für die Erstellung weiterer und umfangreicherer Bewegungsanimationen, Zugang zu einem Motion-Capture-System zu erhalten. Diese Bewegungsdaten können die per Hand erstellten Animationen ersetzen und das System mit mehr Realismus weiter aufwerten.

H-Anim Chart

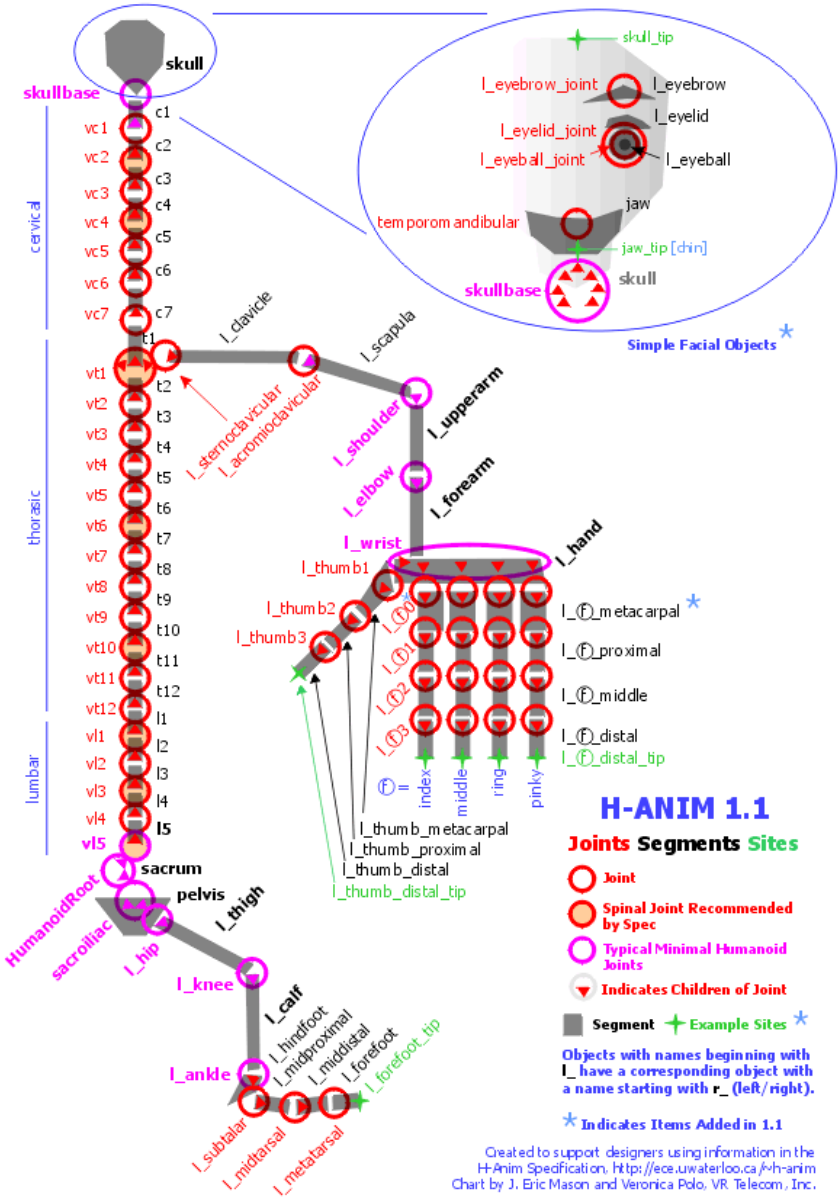

Abbildung: Überblick aller H-Anim-Joints [aus HANIM, Abschnitt D.4]

Abbildungsverzeichnis

Abkürzungsverzeichnis

API	*Application Programming Interface*
ASCII	*American Standard Code for Information Interchange*
DHTML	*Dynamic Hyper Text Markup Language*
EAI	*External Authoring Interface*
GUI	*Graphical User Interface*
HTML	*Hyper Text Markup Language*
IK	*Inverse Kinematik*
JDK	*Java Developer Kit*
JRE	*Java Runtime Environment*
JVM	*Java Virtual Machine*
LoA	*Level of Articulation*
NURBS	*Non Uniform Rational B-Spline*
OI	*OpenInventor*
PHP	*PHP: Hypertext Preprocessor* (rekursives Akronym)
SGI	*Silicon Graphics, Inc.*
VAG	*VRML Architecture Group*
VRML	*Virtual Reality Modeling Language*
WWW	*World Wide Web*

Literaturverzeichnis

[Amm97] Amman, Eckhard: *Programmierung animierter Welten*, Bonn: International Thomson Publishing GmbH, 1997

[Bah00] Bahrmann, Gudrun et al: *Kabale und Liebe - Begleitmaterial zur Inszenierung*, Berlin: Carrousel Theater, 2000, gebundene Broschüre zum Stück von Friedrich Schiller

[Däß98] Däßler, Rolf; Palm, Hartmut: *Virtuelle Informationsräume mit VRML*, Heidelberg: dpunkt, Verlag für digitale Technologie, 1998

[Däß99] Däßler, Rolf: *VRML*, Kaarst: bhv Verlag, 1999

[Die97] Diehl, Stephan: *Java & Co*, Addison Wesley, 1997

[Die01] Diehl, Stephan: *Distributed virtual worlds*, Springer, 2001

[Fla97] Flanagan, David: *JavaScript: The Definitive Guide*, O'Reilly, 1997

[Has97] Hase, Hans-Lothar: *Dynamische virtuelle Welten mit VRML 2.0*, Heidelberg: dpunkt, Verlag für digitale Technologie, 1997

[Köt01] Kötter, Engelbert et al.: *Literaturverfilmung*, Berlin: Cornelsen Verlag, 2001

[Läg00] Läge, Melanie; Suicmez, Attila S.: *Skript Virtuelle Realität*, Berlin: Logos Verlag, 2000

[LuZ02] Lu, Ruqian; Zhang, Songmao: *Automatic Generation of Computer Animation, Using AI for Movie Animation*, Springer, 2002

[Mat96] Matsuba, Stephen N.; Roehl, Bernie: *VRML*, München: Markt & Technik Verlag, 1996

[Nie97] Niessner, Andreas: *VRML 2.0 - Virtuelle Welten dreidimensional modellieren*, München: Richard Pflaum Verlag, 1997

[Pai98] Pai, Rajesh; Govil-Pai, Shalini: *Learning Computer Graphics*, Springer, 1998

[Rie01] Riegler, Alexander et al.: *Virtual reality: Cognitive Foundations, Technological Issues & Philosophical Implications*, Lang, 2001

[Rog01] Rogers, David F.: *An introduction to NURBS: with historical perspective*, San Fransisco: Morgan Kaufmann Publishers, 2001

[Sch02] Schmidt, Klaus: *PHP 4 - Tutorial und Referenz*, Böblingen: C & L Computer und Literaturverlag, 2002

[Sch54] Schiller, Friedrich: *Kabale und Liebe*, In: NAME, VORNAME (Hg.): *Gesammelte Werke - zweiter Teil* Berlin: Aufbau-Verlag, 1954, S. 315 ff.

[Stu98] Stuhrmann, Christoph et al.: *HTML 4*, Düsseldorf: Data Becker, 1998

[The00] Theis, Thomas: *PHP 4*, Bonn: Galileo Press GmbH, 2000

[Wen01] Wenz, Christian: *JavaScript*, Bonn: Galileo Press GmbH, 2001

[Vac96] Vacca, John R.: *VRML - Bringing Virtual Reality to the Internet* AP Professional / Academic Press Inc., 1996

[Wen98] Wendt, Volker: *3D Studio Max R2.x*, Kaarst: bhv Verlag, 1998, S. 271-277

Onlinequellen

[HANIM] http://h-anim.org/Specifications/H-Anim1.1/
The H-Anim Consortium:
Specification for a Standard Humanoid v1.1, Stand Juli 2002

[Web3D A] http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm
The Web 3D Consortium:
The Virtual Reality Modeling Language, Part 1:
Base functionality and text encoding for VRML, Stand Juni 2002

[Web3D B] http://www.web3d.org/technicalinfo/specifications/eai/index.html
The Web 3D Consortium:
The Virtual Reality Modeling Language, Part 2:
Base functionality and bindings for the
VRML External Authoring Interface, Stand August 2002

[PULSE] http://www.pulse3d.com/solutions/index.asp
Pulse, Stand September 2002

[VRML 1.0] http://www.vrml.org/VRML1.0/vrml10c.html
Bell, Gavin et al:
The Virtual Reality Modeling Language:
Version 1.0 Specification, Stand Juni 2002

Danksagung

Ich möchte allen Personen danken, die mich während der Bearbeitungszeit dieser Diplomarbeit unterstützt haben: Prof. Jung und Prof. Naumann, die immer Zeit für ein Gespräch hatten; Annika und Frau Ritter für das Korrekturlesen. Dank gebührt auch Sebastian, Hagen und Thomas für ihre Hilfe beim Test des Prototypen.

Eigenständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Steffen Rademacher,
Berlin, 18.11.2002