Universität Leipzig

Fakultät für Mathematik und Informatik

Institut für Informatik

Spezifikation und Implementation virtueller Räume in einer Java−basierten Verteilungsumgebung im WWW

DIPLOMARBEIT

Vorgelegt von:
Andreas Müller

Betreut durch:
Prof. Dr. K. Irmscher
Dipl. Inf. H. Schulze

Leipzig, Februar 2000

Inhaltsverzeichnis

Abbildungsverzeichnis

Verwendete Abkürzungen

ASCII	American Standard Code for Information Interchange
API	Application Programming Interface
CAD	Computer Aided Design
EAI	External Authoring Interface
GNU	GNU's Not UNIX! Name der Free Software Foundation
GPL	GNU Public Licence
HPUX	Hewlet Packard Unix
HTML	Hypertext Markup Language
IRC	Internet Relay Chat
JDK	Java Development Kit
LOD	Level of Detail Node
MS	Microsoft
MUtech	Multi−User Technologie
PROTO	Prototype
SAI	Script Authoring Interface
SGI	Silicon Graphics International
URL	Uniform Resource Locator
UTC	Weltzeit
UTF−8	Unitext Code
VOP	Virtual Objects Package
VRML	Virtual Modelling Language

Aufgabenstellung

Ziel der Diplomarbeit ist die Spezifikation und Implementation von virtuellen Räumen in einer Java−basierten Verteilungsumgebung im WWW. Im Rahmen der Arbeit sollen mittels Java und VRML unter Nutzung des bereitgestellten API (Application Programming Interface) und EAI (External Authoring Interface) virtuelle Räume definiert, spezifiziert und implementiert werden, die den Benutzern einen virtuellen Arbeitsraum zur Verfügung stellen und es ihnen ermöglichen mit anderen Nutzern zu interagieren. Als erste Lösung ist ein virtueller Chatroom bereitzustellen, der den Räumen 5−01, 5−05 und 5−26 des Universitäts−Hauptgebäudes nachempfunden ist. Dabei sind die Kommunikationsschnittstellen und die virtuelle Darstellung der Benutzer (Avatare) in einer Mehrbenutzerwelt zu erarbeiten.
Für die zugrunde liegende Client/Server Architektur ist ein Vergleich mit herkömmlichen Java−Socket basierten Lösungen (VNET) durchzuführen. Durch entsprechende Schnittstellengestaltung bzw. −anpassung ist eine multivalent einsetzbare Lösung anzustreben. Die Implementation sollte auch Auskunft über die mit dem System durchgeführten Aktivitäten geben (Logging).

Selbständigkeitserklärung:

Hiermit versichere ich, das ich die vorliegende Diplomarbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Leipzig, Februar 2000

1 Einleitung

1.1 Motivation

VRML, die Abkürzung von Virtual Modelling Language, ist das Ergebnis einer bis heute noch nicht abgeschlossenen Reihe von Vorlagen und Diskussionspapieren (sog. Papers) zur Realisierung eines Standards für die Darstellung dreidimensionaler Welten im World Wide Web. Dabei stand und steht nicht die möglichst realistische Darstellung von Objekten im Vordergrund, so wie dies zum Beispiel bei Computerspielen oder CAD Anwendungen der Fall ist, vielmehr soll mittels einer übersichtlichen Struktur und einer ganzen Reihe von Schnittstellen die Möglichkeit gegeben werden, den klassischen Inhalten des WWW, wie Text und Bild, eine dritte Dimension hinzuzufügen zu können, welche sich bezüglich der verwendeten Bandbreiten an die vorhandenen Gegebenheiten anpassen läßt.

An der Entwicklung von VRML sind sowohl kommerzielle Firmen, wie Silicon Graphics und Sony, als auch eine große Anzahl freiwillig mitarbeitender Computerspezialisten beteiligt. Die Entwicklung vollzieht sich dabei für jedermann sichtbar innerhalb von Arbeitsgruppen, welche die Ergebnisse ihrer Arbeit auf den Seiten des Web3D Consortiums (früher VRML Consortiums) bereitstellen. Es gibt Mailing−Listen in denen jeder VRML Interessierte die aktuellen Entwicklungen mitverfolgen und durch eigene Anfragen und Anregungen mitgestalten kann.

Von Anfang an wurde dabei auf eine schrittweise Entwicklung des Standards Wert gelegt, um innerhalb eines vernünftigen Zeitplanes zu bleiben und notwendige Modifikationen besser einfügen zu können. Während VRML Version 1.0, welche 1994 vorgestellt wurde, sich lediglich auf die Darstellung dreidimensionaler Objekte beschränkte, sind seit der aktuellen Version 2.0 bereits umfangreiche Interaktionen zwischen Objekten der virtuellen Welt und dem Benutzer möglich. Angefangen von animierten Objekten, über Sensoren die Ereignisse aufnehmen und weiterleiten, bis zur Unterstützung von Multimediaelementen wie Sound oder Film ist es möglich geworden, virtuelle Szenen nicht nur auf visueller Seite realistischer zu gestalten. Durch die Einbindung von Schnittstellen für höhere Programmiersprachen wie C/C++ und Java können die Möglichkeiten des Standards fast beliebig erweitert werden, so sind neben den klassischen Anwendungsgebieten, wie der Darstellung Objekten und Szenen in 3D, auch solche wie zum Beispiel Landschaftsgeneratoren, Generierung dreidimensionaler Diagramme aus Datenmaterialien, oder grafische Interfaces in 3D welche auf gebräuchliche Datenbanksysteme zugreifen können, möglich.

Der nächste Schritt in der Entwicklung von VRML stellen die Erweiterungen zur Implementation von Mehrbenutzerwelten dar, denn bis dato war der Besucher einer virtuellen Welt allein. Schon heute gibt es eine ganze Reihe von Mehrbenutzerwelten im Internet, welche durch die Verwendung der externen Schnittstellen, welche VRML für andere Hochsprachen bietet, realisiert werden können. Einige dieser Welten werden bereits mit Erfolg kommerziell genutzt, als virtueller

Treffpunkt für Unterhaltungen oder Diskussionen, virtuelle Einkaufszentren oder zur Demonstration von Produkten auf virtuellen Messen. Viele dieser Welten sind aber auch völlig uneigennützig entstanden, aus Freude, Spieltrieb, um die Grenzen des Systems auszutesten oder im verstärkten Maße auch als Projekte in Schulen und Ausbildungszentren.

Die vorliegende Diplomarbeit beschäftigt sich mit den Einsatzmöglichkeiten des aktuellen Standards 2.0 von VRML als Grundlage einer virtuellen und vernetzten Welt unter Zuhilfenahme der Programmiersprache Java, welche die fehlenden Komponenten zu Vernetzung und Nutzerverwaltung bereitstellt. Dazu werden zunächst die wichtigsten Elemente der VRML 2.0 Spezifikation, sofern diese für die vorliegende Arbeit relevant sind, erläutert, sowie bereits existierende Konzepte für Multiuserwelten vorgestellt und diskutiert. Neben einer Vorstellung der gebräuchlichsten Werkzeuge und Browser wird weiterhin eine praktische Implementation einer virtuellen Umgebung beschrieben, welche speziell für den Praktikumsbetrieb an der Universität Leipzig entwickelt wurde, und außerdem Vorschläge zur Gestaltung eines VRML Praktikums innerhalb der Informatikausbildung enthält. Abschließend werden die praktischen Ergebnisse erläutert und ein Ausblick auf künftige Erweiterungen des Projekts zum einen, aber auch der Entwicklung von VRML zum anderen gegeben.

1.2 Arbeitsablauf

Zum Einsatz kommt ein bereits bestehendes, auf Java basierendes VRML Chat−system (VNET). Dieses ist durch die bereits genannten Applikationen zu ergän−zen. Es ergeben sich folgende Teilbereiche:

- Entwicklung eines VRML−Prototypen welcher eine Standardschnittstelle für alle Objekte der virtuellen Welt zur Verfügung stellt sowie die Weiterleitung beliebig zu definierender Ereignisse an die Java−basierten Schnittstellen ermöglicht.
- Ebenso ist eine Java−basierte Schnittstelle/Applet zu entwickeln und imple−mentieren welches die vom oben genannten. Prototypen bereitgestellten Mög−lichkeiten umsetzt und die Vernetzung der Welten übernimmt. Außerdem müssen Modifikationen in der Struktur der virtuellen Welt erkannt und auto−matisch eingebunden werden können.
- Es kommt ein spezieller Server zum Einsatz welcher die ankommenden Ereig−nisse speichert, filtert und entsprechend weiterleitet, zusätzlich ist außerdem eine einfache Benutzerverwaltung vorgesehen. Auch dieser Server soll in Java implementiert werden.

Das Ergebnis sollte folgende Eigenschaften aufweisen:

- Mit einem normalen HTML Browser mit VRML Plugin abrufbar
- Einzel− und Mehrbenutzerbetrieb
- Vernetzung der Clients mittels Java Sockets
- Enthält ein Applet zur textbasierenden Kommunikation (Java−Chat)
- Die Benutzer treten in Form von Avataren in der virtuellen Welt auf
- Eintretende Ereignisse (Bewegung von Avataren, Öffnen von Türen usw.) sol−len auf alle angeschlossenen Clients übermittelt werden.
- Einbindung von externen Applikationen bei Anwendungen für die es im momentanen VRML Standard noch keine Unterstützung gibt (Video/Audio−Ströme)
- Bereitstellung von Standard−Schnittstellen zu einfachen Erweiterbarkeit des Systems
- Optimierung der Geschwindigkeit des gesamten Systems für langsamere Rechner
- Einsatz im Praktikumsbetrieb der Universität

- Als 1. Lösung ist ein virtueller Chatroom bereitzustellen der den Räumen 5−01,5−05 und 5−26 nachempfunden ist.

Im Verlauf der Durchführung der Diplomarbeit wurden verschiedene bereits vorhandene Konzepte zum Aufbau von virtuellen Welten untersucht, verglichen und teilweise bewertet, was in den folgenden Abschnitten beschrieben wird. Zuvor jedoch wird zunächst der Aufbau und Funktionsweise der verwendeten Modellierungssprache erläutert und auf einige Besonderheiten, welche für das vorliegende Projekt interessant sind, eingegangen.

1.3 Die Struktur dieses Dokuments

Kapitel 1 enthält neben einem Überblick über die Thematik die Beschreibung der Aufgabenstellungen sowie eine Aufstellung der durchzuführenden Arbeitsschritte.

Kapitel 2 enthält eine Einführung über VRML als Beschreibungssprache für virtuelle Welten und eine Erläuterung des Standards

In **Kapitel 3** werden verschiedene bereits existierende Standards für verteilte virtuelle Welten unter VRML vorgestellt und diskutiert

Kapitel 4 beschäftigt sich mit den gebräuchlichsten VRML Browsern und untersucht deren Verwendbarkeit im vorliegenden Projekt.

Kapitel 5 beschäftigt sich mit dem External Authoring Interface, der wichtigsten Schnittstelle zwischen VRML Browser und Java Applets.

In **Kapitel 6 und 7** schließlich wird die Umsetzung und Implementation einer für den Praktikumsbetrieb an der Universität entwickelten Lösung einer verteilten virtuellen Umgebung erläutert und Vorschläge zur Gestaltung eines entsprechenden Praktikums gegeben.

2 VRML als Grundlage für virtuelle Welten im Inter−net

Sucht man im WWW nach Definitionen für den Begriff der virtuellen Welten (bzw. Virtual Reality in Englisch) stößt man häufig auf zwei geläufige Definitio−nen welche oftmals auch gemeinsam angegeben werden:

Definition I

»Virtual Reality: A computer system used to create an artificial world in which the user has the impression of being in that world and with the ability to navigate through the world and manipulate objects in the world.«

− C. Manetta and R. Blade in "Glossary of Virtual Reality Terminology" in the International Journal of Virtual Reality, Vol.1 Nr.2 1995.

Definition II

»Virtual Reality allows you to explore a computer generated world by actually being in it«

− B. Sherman and P. Judkins (1992) "Glimpses of Heaven, Visions of Hell: Virtual Reality and its implications" (Hodder and Stoughton: London).

Die konkrete Definition ist also abhängig vom Standpunkt des Betrachters und dessen Bedürfnissen: Einerseits steht die Erzeugung einer fiktiven bzw. der Rea−lität nachempfundenen, immer aber künstlichen Welt im Vordergrund, anderer−seits der eigentliche Eindruck des Benutzers selbst ein Teil dieser Welt zu sein. Letzterer Aspekt hat in den letzten Jahren wesentlich an Bedeutung gewonnen, nachdem sich die Geschwindigkeit und Qualität grafischer Simulationen und Darstellungen immer mehr verbessert hat. Denn trotz aller Dreidimensionalität produziert der flache Bildschirm nach wie vor nur zweidimensionale Bilder, einen Ausweg liefern sog. 3D Brillen und Masken um diese gewünschten Effekte zu ermöglichen. Für das Internet sind diese Erweiterung aufgrund der hohen zu übertragenden Datenmengen noch nicht anwendbar, außerdem ist ein solcher Effekt kompletter Illusion nicht immer und überall sinnvoll bzw. erwünscht. Ingesamt kann man sagen das dies aber eher ein Problem der verwendeten Hard− und Browsersoftware ist.

VRML als Modelierungssprache wird also hauptsächlich Definition I gerecht: Erzeugung und Darstellung künstlicher Welten und die Bewegung des Anwenders in diesen Welten. In Bezug auf Realitänsnähe ist VRML ein Kompromiss, ange−paßt auf die aktuellen Gegebenheiten die uns das heutige Internet in Bezug auf Bandbreite und Geschwindigkeit liefert. Das heißt, einfach aufgebaute, gering strukturierte Welten ermöglichen ein schnelles Laden und flüssige Ausführung auch auf weniger leistungsfähigen Systemen, andererseits ist es auch möglich

detailierte, stark strukturierte Szenarien zu erzeugen welche natürlich höhere Ansprüche an Bandbreite und Rechnerleistung stellen. Aber auch hier bietet die Sprache durch Optimierung die Möglichkeit sowohl die Ladezeiten zu verkürzen (Verwendung von komprimierten Dateien) als auch den Aufwand zur berechnung der Szenarien zu veringern (mehr dazu im Abschnitt 6.3).

Das vorliegende Kapitel soll einen möglichst allgemeinen Überblick über die Modelliersprache VRML als 3D Standart im WWW geben, wobei allerdings auf eine detailierte Beschreibung der einzelnen Komponenten verzichtet wird, da diese in vielfältiger Form sowohl im WWW als auch in Buchform zu finden sind (siehe Literatur− und Linkverzeichnis im Anhang). Auf einige spezielle Details wie das VRML−EAI, welches im vorliegenden Projekt eine besondere Rolle spielt, wird in einem gesonderten Kapitel Raum gewidmet. An dieser Stelle sei auf die Seiten des Web3D Consortiums[1] hingewiesen (www.web3d.org) wel−che sowohl alle aktuellen Standards enthält als auch die neusten Entwürfe zu zukünftigen Standards. Weiterhin sind zahlreiche Dokumente über die Tätigkeiten der verschiedenen Arbeitsgruppen sowie Berichte von Symposien und Konferen−zen vorhanden.

2.1 VRML−Entwicklung

Die Virtual Modelling Language wurde im Ergebnis der internationalen WWW Konferenz im Mai 1994 in Cern entwickelt, auf der über die Einführung eines allgemeinen plattformabhängigen Standards für die Darstellung von 3D Objekten im WWW diskutiert wurde. Dabei wird der Begriff VRML für Virtual Markup Language geprägt, später wird das Markup in Modelling umgewandelt, sowie eine Mailing Liste eingerichtet. Um den abzusehenden, langwierigen Prozess der Ent−wicklung eines umfassenden Standards zu vermeiden, beschließt die VRML Gemeinschaft noch im gleichen Jahr ein 3−Stufenprogramm für die Entwicklung von VRML:

1. Definierung von Aussehen (appearance): Wie Objekte in Cyberspace erschei−nen
2. Definitierung von Verhalten (behavior): Wie Objekte im Cyberspace bewegt werden können
3. Definierung von Verteilung (distribution): Wie verteilte Umgebungen im Cyberspace funktionieren

Auf Grundlage dieses Programs sind bisher folgende Versionen des VRML−Standards entstanden:

> **VRML 1.0** auf Grundlage von Open Inventor (Silicon Graphics) wird im Oktober 1994 auf der zweiten WWW Konferenz in Chicago vorgestellt. Einem Aufruf an alle Interessenten auf der SIGRAPH 95 (August) folgte die Gründung der VAG (VRML Architekture Group). Aufgabe dieser Gruppe ist hauptsächlich die Koordinierung des Standardisierungsprozesses. Von ihr wird auch die Spezifikation für VRML 2.0 aufgestellt.

1 Das Web3D Consortium koordiniert die weltweite Entwicklung von VRML und betreibt mehrere Arbeitsgruppen welche verschiedene Teilbereiche des Standards bearbeiten

VRML 1.1 welche Version 1.0 durch Interaktionen ergänzen soll, kommt nicht mehr zum Zuge, so kommt es im Januar 1996 zur öffentlichen Abgabe von Vorschlägen zu:

VRML 2.0: Aus den Vorschlägen von Apple, GMD, IBM Japan, Microsoft und SGI wird nach Abstimmung im Internet schließlich der Vorschlag von SGI angenommen. Arbeitstitel: "Moving Worlds". Zunächst werden aber die Festlegung einer allgemeinen Schnittstelle zwischen VRML und Scriptsprachen aus der Spezifikation herausgenommen, die später als Anhänge für Java/JavaScript erscheinen. Weitere wichtige Ereignisse in der Entwicklung von VRML 2.0 waren außerdem:

04.08.1996 Abschluß der Spezifikation

6.08.1996 Vorstellung auf der SIGGRAPH 96 in New Orleans und Gründung des VRML Consortium. VRML 97 ist nun die inoffizielle Bezeichnung für den internationalen Standard ISO/IEC 14772−1:1997 und gilt seit Dezember 97 als Nachfolger bzw. Endprodukt von VRML 2.0.

1996 wurden dann die unter dem Arbeitstitel „*Moving Worlds* eingereichten Entwurfsvorlagen von SGI und Sony als VRML 2.0 Spezifikation bestätigt.

Die wichtigsten Unterschiede bzw. Erweiterungen zu VRML 1.0 sind nach [5]:

- Unterstützung von Audio, Videotexturen, Nebel, Panoramahintergründe und Landschaftsstrukturen
- Objektineraktionen durch Kollisionserkennung, Proximity− und andere Sensor Nodes
- Animationen sowie Einbindung höherer Programmiersprachen wie Java/JavaScript welche das Verhalten von Objekten beeinflussen können.
- Prototyping von komplexen Objekten so das diese mehrfach verwendet werden können[2].

2.2 Aufbau von VRML

Nach [5] ist VRML 1.0 eine szenenbeschreibende Sprache dessen Dateiformat auf dem Silicon Graphics Open Ventor Toolkit basiert. Es ist weder eine echte Programmiersprache wie C++ oder Java, noch eine Seitenbeschreibungsprache (Markup Language) wie HTML. Vielmehr ist es eine Modellierungssprache welche 3D Szenerien beschreibt. Technisch gesehen besteht eine VRML Welt lediglich aus einer Liste von Objekten welche sich auch gegenseitig beeinflussen können.

2 Wobei Aussehen und Eigenschaften durch Parameterübergabe beeinflußt werden können

2.2.1 Dokumentstruktur

Jedes VRML Dokument ist eine im ASCII Kode abgespeicherte Textdatei um den Kode selbst plattformübergreifend bereitstellen zu können. Wärend unter VRML 1.0 nur der Zeichensatz des 7−Bit ASCII−Standards dargestellt werden konnte bildet seit der Spezifikation 2.0 der UTF−8 Standard die Grundlage für den VRML Quelltext. Um dem Browser die im Quelltext verwendete VRML−Version mitzuteilen ist am Anfang einer jeden VRML Datei zwingend ein Vermerk in der folgenden Syntax anzubringen:

```
#VRML V1.0 ascii
```

bzw.

```
#VRML V2.0 utf8
```

Fehlt dieser Eintrag geben die meisten VRML Browser eine Fehlermeldung der Form *Dokument is empty* aus.

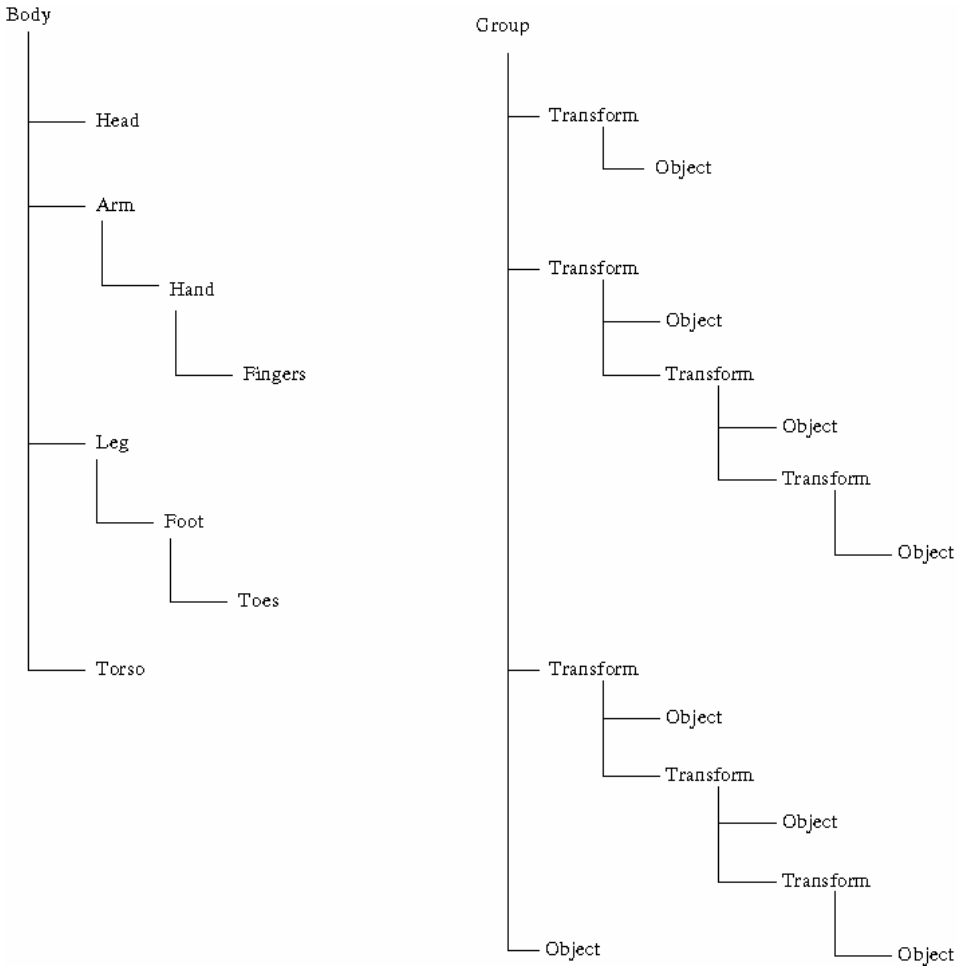

Abbildung 1: Scenegraph eines VRML−Dokuments

Grundlage jedes VRML Dokuments ist der sogenannte *Scenegraph* (siehe Abbil−dung 1). Ein Dokument setzt sich aus verschiedenen Knotenpunkten (Nodes)

zusammen welche hierarchisch strukturiert werden können und in ihrer Gesamtheit den Scenegraph bilden. Übergeordnete Knoten beeinflussen die darundergeordneten Knoten (children) direkt.

2.2.2 VRML Begriffsbestimmung

In VRML werden hauptsächlich zwei Datenstrukturen verwendet: Nodes (Knoten) und Fields (Felder)[3]. Zitat aus [4]:

> *A node is the equivalent of a Java class. A field is (almost) the equivalent of a variable and a Java method rolled into one.*

Nodes
Nodes beinhalten Felder welche die Eigenschaften dieses Nodes bestimmen. Ein Node ist die kleinste einzelne Grundstruktur einer jeden VRML−Datei. Ähnlich wie in Java (aber nicht wie z.B. in C/C++) kann ein Node für sich allein stehen, ein Feld aber nicht. VRML 2.0 enthält 54 Nodes welche in 9 Gruppen nach ihrer Funktion eingeordnet werden:

Grouping Nodes: Enthält eine Liste von untergeordneten Nodes (Kindknoten= *children nodes*) welche die Eigenschaften des Group−Nodes übernehmen. Jeder *group node* definiert ein eigenes Koordinatensysten zu seinen Kindknoten welches relativ zu seinem eigenen Koordinatensystem ist. Die Kindknoten selbst können ebenfalls Instanzen von *grouping nodes* sein und somit eine Hirarchie bilden.

Special Groups sind *grouping nodes* welche zusätzliche spezielle Eigenschaften aufweisen, eingeschlossen das Selektieren eines von vielen Kindknoten nach einem vorgegebenen Parameter (LOD, Switch) oder das dynamische Laden von Nodes aus einer externen Datei (Inline).

Common Nodes: Nodes welche normale Objekte und Parameter beinhalten

Sensors: Nodes welche es dem Benutzer ermöglichen interaktiv, zum Beispiel durch anklicken von Objekten, mit der virtuellen Welt zu kommunizieren. *Sensor Nodes* reagieren auf Interaktionen mit geometrischen Objekten, der Bewegung des Benutzers durch die Welt oder in Abhängigkeit von zeitlichen Abständen.

Geometry Nodes: Enthalten die mathematische Beschreibung von dreidimensionalen Punkten, Linien, Oberflächen, Textzeilen und Körpern.

Geometry Properties: Einige *geometry nodes* definieren Eigenschaften wie Koordinaten, Farbe, Texturkoordinaten und Normale als geometrische Eigenschaften und werden daher gesondert eingeordnet.

---

3 Im Folgenden werden nur noch die englischen Bezeichnungen verwendet da diese als Teil der Sprache auch in den Quelltexten auftauchen

Appearance Nodes: Spezifizieren das Aussehen von Körpern (*Shapes*) mittels Beschreibung von Material und Textureigenschaften.

Interpolators: Entwickelt für zeit− und ereignisgesteuerte Animationen stellen diese Nodes eine endliche Anzahl von Parametern (Koordinaten, Winkel etc.) zur Verfügung welche in einem definierten Intervall ausgeführt werden. Zwischenparameter werden dabei durch den Node interpoliert.

Bindable nodes: Nodes welche viele Instanzen innerhalb eine *Scenegraps* besitzen, wobei aber nur jeweils eine Instanz zu einem Zeitpunkt aktiv ist. Beispiele sind *Viewpoint* und *Fog*.

Da eine komplette Aufzählung der Nodes den Rahmen dieser Diplomarbeit sprengen würde möchte ich hier auf Dokumentation in der VRML 2.0 Spezifikation verweisen. Diese befindet sich u.a auch auf der zu dieser Diplomarbeit angelegten Homepage. Nodes selber können in vielfältiger Form auftreten, es gibt Nodes zur:

- Darstellung einfacher geometrischer Objekte (Shapes)
- Darstellung komplexer geometrischer Oberfächen (elevationGrids)
- Transformierung von Objekten in andere Koordinatensysteme (Transform)
- Zusammenfassung mehrer Objekte zu einer Gruppe (grouping nodes)
- Sensoren (ab VRML 2.0: TimeSensor, ProximitySensor und andere)
- Animation von Objekten (Interpolatoren)
- Prototypen (selbstdefierte und wiederverwendbare Nodes)
- Einbindung von Java und JavaScript bzw. anderen Hochsprachen

Der prinzipielle Aufbau eines Nodes is dabei:

```
[DEF objectname] objectType {[fields][children]}
```

wobei *objectType* der Name des Nodes ist. *Fields* sind Parameter welche an den Node übergeben werden können, *Children* sind in der Hierarchie untergeordnete Nodes. Jedem Node kann auch eine Bezeichnung zugewiesen werden, welche mit dem Vorsatz DEF definiert wird.

2.2.3 Koordinatensysteme

Ein wichtiges Element von VRML als Beschreibungssprache für dreidimensionale Welten ist die umfassende Syntax welche für die Definition des Standortes von Objekten und deren Größe verwendet wird. Dabei gibt es kein einheitliches, gro−ßes Koordinatensystem, stattdessen kann gegebenenfalls jedes einzelne Objekt ein eigenes Koordinatensystem zugewiesen werden. Der Aufbau der Koordinatensy−steme entspricht dabei dem der in der dreidimensionalen Geometrie verwendenten Systems der nach der Rechtehandregel angeordneten Koordinatenlachsen x,y und z:

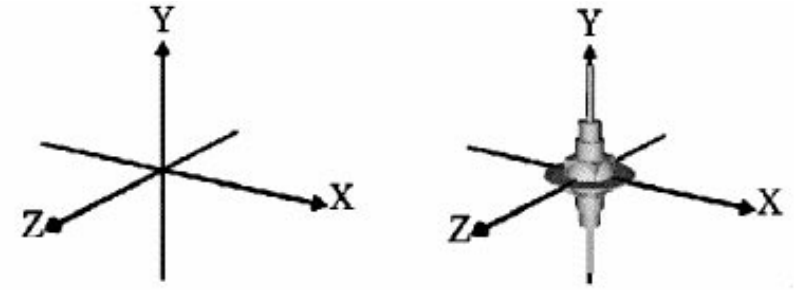

Abbildung 2: XYZ−Koordinatensystem von VRML

Die Festlegung der Parameter erfolgt im Transform Knoten mittels ingesamt 17 Parameter:

```
Transform {
    center 0.0 0.0 0.0
    translation 1.0 1.0 1.0
    rotation 0 1 0 1.54
    scale 1.0 1.0 1.0
    scaleOrientation 0 0 0 1 0
    children [ ]
}
```

center Verschiebung des lokalen Koordinatensystems, dieser Punkt wird als Koordinatenursprung bei der Skalierung und Rotation verwendet

translation ist die Verschiebung des/der Objekte(s) gegenüber dem übergeordne−ten Koordinatensystem

rotation ist ein dreidimensionaler Vektor welcher die Neigung des/der Objekte(s) auf einer Achse des übergeordneten Koordinatensystems festlegt.

scale Skalierung des lokalen Koordinatensystems, wobei die Skalierung ungleichförmig ist (unterschiedliche Skalierungsfaktoren in jeder Richtung)

scaleOrientation ist ebenfalls ein dreidimensionaler Vektor welcher sich aber nur unmittelbar auf die Skalierung auswirkt

Als Basis−Koordinatensystem für das gesamte VRML Dokument wird ein metri−sches[4] System mit dem Koordinatenursprung [0.0 0.0 0.0] verwendet.

2.2.4 Prototypen

Der Erfolg von höheren Programmiersprachen resultiert unter anderem auch dar−auf das eine einfache Sprache ohne irgendwelche proprietären Erweiterungen durch Verwendung und Wiederverwendung von Bibliotheken, welche in der sel−ben Sprache geschrieben wurden, erweitert werden kann. Das selbe Ziel verfolgte auch die einführung von Prototypen in VRML 2.0, nämlich die einfache, modu−lare Erweiterbarkeit innerhalb der Grenzen von VRML. Ähnlich wie in höheren Programmiersprachen kann ein Prototype (*PROTO*) als Black Box betrachtet werden welche ohne Kenntnis ihres inneren Aufbaus überall in der Szene ver−wendet werden kann. Der prinzipielle Aufbau eines Prototypen in VRML sieht wie folgend aus:

```
PROTO prototypname
        [       #interface
          eventIn's   name   value
                     ...
          eventOut's  name   value
                     ...
          exposedFields name  value
                     ...
        ]
# scene
{
  nodes
  prototypes
  routes
}
```

Innerhalb der Szene können die Felder und Ereignisse den Knoten innerhalb der Prototypendefinition zugewiesen werden:

```
nodename {name IS new_name}
```

Dabei dürfen der ursprüngliche und neue Name durchaus identisch sein ohne das es zu Konflikten kommt. Und auch hier gilt es zu beachten das alle Variablen und Feldnamen case−sensitiv sind, andernfalls gibt der VRML Browser eine Fehler−meldung der Form *„EventIn Name not found“* aus.

4 dieser metrische Bezug wird auch bei anderen Nodes verwendet, wie zum Beispiel im Shape Node zur Festlegung der Größe (size) eines Quaders (box)

Prototypen selber können sowohl im eigentlichen VRML Hauptdokument als auch extern in einer Bibliothek deklariert werden, bei letzterem handelt es sich dann um sogenannte externe Prototypen (*Externprotos)*. Die Instanziierung erfolgt ähnlich wie in höheren Programmiersprachen:

```
[DEF bezeichnung] prototypename {
                    parameter 1
                    parameter 2
                          ....
                  }
```

Die Bezeichnung der Instanz ist optional, ebenso können, anders als in höheren Programmiersprachen, weniger Parameter übergeben werden als ursprünglich im Prototypen definiert wurden. Der VRML Browser setzt dann an den fehlenden Stellen eigene oder die im Protokopf vorgegebenen Standardwerte ein. Ist ein Prototype in einer externen Bibliothek abgelegt worden muß er vorher am Beginn des VRML Dokuments als Externproto deklariert werden, dies geschieht in fol−gender Weise:

```
EXTERNPROTO Beispiel[]"http://.../protos.wrl#Beispiel"
```

Die Sammlung von Prototypen in einer Bibliothek hat den Vorteil das nur ein einziger Netzzugriff erforderlich ist um eine größere Anzahl von vordefinierten Objekten laden zu können.

2.3 VRML und Java/JavaScript

Obwohl VRML prinzipiell jede höhere Programmiersprache unterstützt, welche die entsprechenden Schnittstellen und Klassen bereitstellen, hat sich Java bzw. JavaScript zur externen Steuerung von VRML durchgesetzt. Innerhalb des URL−Nodes kann dabei JavaScript direkt implementiert werden, oder die URL einer Java Klasse angegeben werden. Das folgende Beispiel (aus [12]) zeigt die Imple−mentation eines Prototypen, welcher beim anklicken die Farbe wechselt und beim loslassen der Maustaste wieder in die Ursprungsfarbe zurückkehrt. Zum Einsatz kommt dabei ein *TouchSensor,* welcher bei gedrückter linker Maustaste den Wert *isActive=TRUE* liefert, bei losgelassener Taste den Wert *isActive=FALSE*:

```
PROTO Highlighter [
               eventIn SFBool isActive
               eventOut SFColor colour
               field SFColor activeColour 1 1 0
               field SFColor inactiveColour 0 0 1
     ]
```

```
{
Script {
          eventIn SFBool isActive IS isActive
          eventOut SFColor colour IS colour
          field SFColor activeColour IS activeColour
          field SFColor inactiveColour IS inactiveColour
          url [
             "vrmlscript:
                   function isActive(eventValue) {
                         if (eventValue == true)
                               colour = activeColour;
                         else
                               colour = inactiveColour;
                   }",
             "Highlight.class"
          ]
          }
}
```

Bemerkung: Die Bezeichnung *vrmlscript* ist in VRML 2.0 durch *javascript* ersetzt worden, wird aber aus Gründen der Abwärtskompatibilität weiterhin unterstützt.

Die entsprechende Implementation als Java−Klasse importiert zunächst die VRML Klassen gemäß der Spezifikation und definiert anschließend die beiden Methoden *initialize* und *processEvent.*

```
import vrml.*;
import vrml.field.*;
import vrml.node.*;

class Hightlight extends Script {
     private SFColor activeColour;
     private SFColor inactiveColour;
     private SFColor colour;

public void initialize() {
     activeColour = (SFColor)getField("ActiveColour");
     inactiveColour = (SFColor)getField("inactiveColour");
     colour = (SFColor)getEventOut("colour");
}

public void processEvent(Event event) {
     if (event.getName().equals("isActive"))
      if (((ConstSFBool)event.getValue()).getValue()== true)
            colour.setValue(activeColour);
     else
            colour.setValue(inactiveColour);
}
}
```

Der Zugriff auf die VRML−Felder des *SensorNodes* erfolgt durch die Methoden *getField()* und *getValue()*, die genaue Funktion dieser Methoden wird später im Abschnitt 5.3 beschrieben. Gegenüber der relativ einfachen durchführbaren Implementation von JavaScript−Skripten bringt der Einsatz von Java−Klassen Vorteile bei umfangreichen Aufgabenstellungen, außerdem sind Java−Applika−tionen im Gegensatz zu JavaScript in der Lage mit anderen Applikationen zu kommunizieren und auf externe Bibliotheken zuzugreifen.

3 VRML−Browser

3.1 Arten von VRML Browsern

Prinzipiell unterscheidet man bei VRML Browser zwei Arten von Programmen: Entweder ist der Browser als eigenständiges Programm oder als Plugin verfügbar, es gibt auch Browser welche beide Modi beherrschen. Die noch bis vor einigen Jahren vorgenommene Unterscheidung zwischen VRML 1 und 2.0 Browsern kann heutzutage als unnötig betrachtet werden da alle aktuellen Browser beide Versionen unterstützen[5].

3.2 Anforderungen an einen VRML Browser

Nach [1] und [5] sollte ein guter VRML Browser die folgenden Anforderungen erfüllen:

- Vollständige VRML 1.0 und VRML 2.0 Unterstützung

- Flüssige, möglichst ruckelfreie Darstellung bei Bewegung durch die Szene. Dies hängt neben der Leistungsfähigkeit der verwendeten (Grafik−) Hardware und der eingesetzten Grafikroutinen auch von der Art der verwendeten Darstellung ab. So sollte ein guter Browser auch Einstellmöglichkeiten bieten bei denen z.B. auf leistungsschwächeren Systemen auf das Rendern während der Bewegung verzichtet wird und sämtliche Objekte nur als Drahtgittermodelle dargestellt werden.

- Unterstützung verschiedener Navigationsarten. Je nach Art der Umgebung ist es sinnvoll zwischen verschiedenen Arten der Bewegung wechseln zu können. Die wichtigsten Modi dabei sind:

    - **Walk**: Es wird eine Art künstliche Schwerkraft erzeugt die den Betrachter scheinbar am Boden festhält. Ideal zur genaueren und realitätsnahen Erkundung von Szenen.
    - **Fly**: Keine Schwerkraft, der Betrachter kann sich beliebig in jede Richtung bewegen (also auch nach oben und unten). Somit können größere Flächen schneller überwunden werden bzw. sich einen Gesamteindruck aus größerer Höhe gemacht werden (Vogelperspektive)
    - **Examine**: Im Examine Modus steht der Betrachter still und kann die von ihm selektierten Objekte drehen und wenden um sie von allen Seiten betrachten zu können.

- Wahl verschiedener Kameraperspektiven (*Viewpoints*). Der Betrachter soll die Möglichkeit haben vom Ersteller der Szene(n) vorgegebene Standorte aus einer

5 Genauer gesagt handelt es sich dabei meist um reine VRML 2.0 Browser welche intern Dateien, die noch in VRML 1 geschrieben wurden, konvertieren. Ein gutes Beispiel dafür ist vrml1to2.exe in der Windows Version von Cosmoplayer

Liste auswählen und dorthin springen zu können. Das schließt natürlich auch ein das der Betrachter, falls er in einer Szene die Orientierung verloren hat und sich somit wieder an den Ausgangsstandort zurücksetzen kann. Da *Viewpoints* ein Bestandteil des VRML 2.0 Standards sind hat jeder vollständig VRML 2.0 kompatible Browser diese Möglichkeit bereits integriert, allerdings werden nicht von allen Browsern die entsprechenden Schalter auf der Benutzerober−fläche bereitgestellt.

- Die Kollisionskontrolle sollte ein− und ausschaltbar sein. Tatsächlich hatte VRML1.0 standardmäßig noch keine Kollisionserkennung implementiert, wel−ches auch z.B. von [5] bemängelt wurde. Allerdings hatten aber zumindestens die späteren VRML1.0 Browser diese Funktion standardmäßig implementiert.

- HTML und VRML Browser sollten eine Einheit bilden, d.h. der HTML Browser gibt VRML Dokumente an den VRML Browser weiter und umge−kehrt.

- Zur Netzentlastung und schnelleren Ladezeiten sollten unter GNUzip kompri−mierte Dateien,welche die Endung .gz besitzen, automatisch erkannt und ent−packt werden. In der Praxis, und hier vor allem unter MS Windows, kommt es dabei allerdings häufiger zu Verwirrungen, da einige VRML Browser offenbar solche Dateien nur anderen Endung .gz zu erkennen vermögen. Da der Stan−dard eine solche Bezeichnung aber nicht zwingend vorschreibt und sich mitt−lerweile auch noch eine zweite Extension (.wrz) durchgesetzt hat kommt es zu den oben genannten. Irritationen. Prinzipiell kann man sagen das in HTML eingebettete, komprimierte VRML Dateien immer als solche erkannt werden, direkte Links zu solchen Dateien nicht. Der Browser bietet dann meistens an die Datei zu speichern oder ein externes Programm aufzurufen.

3.3 Übersicht über die wichtigsten VRML Browser

3.3.1 Cosmoplayer

Dieser VRML−Browser wurde von Silicon Graphics Inc. (SGI) entwickelt und liegt derzeit in der Version 2.1.1 als Plugin für Netscape Navigator/Communica−tor und Microsofts Internet Explorer unter Windows, MacOS und Irix (nur Net−scape) vor. Cosmoplayer (kurz Cosmo) gilt als VRML Standardbrowser, da alle Komponenten des VRML 2.0 Standards und dessen API/EAI vollständig imple−mentiert sind, was auch darauf zurückzuführen ist das SGI maßgeblich an der Entwicklung von VRML 2.0 mitgewirkt hat. Gegenüber Version 1.0 hat sich die Geschwindigkeit des Browser erheblich verbessert, ab Version 2.1 ist auch die EAI komplett implementiert. Vereinzelt tauchen Probleme beim Starten und Stoppen von Java−Applets auf (Windows) bei denen der Browser komplett abstürzt. Leider hat SGI mittlerweile die Weiterentwicklung dieses Browsers ein−gestellt und den verantwortlichen Mitarbeiterstab entlassen. Die Quellen des Kodes wurden aber als spezielle Open Source Lizenz dem Web3D Consortium zur Verfügung gestellt.

3.3.2 Internet Explorer VRML−Plugin

Dieser von Microsoft herausgegebene Browser ist warscheinlich der direkte Nachfolger des von DimensionX übernommenen Liquid Reality Browsers und wird als Plugin für den Internet Explorer ab Version 4.0 bereitgestellt. VRML 2.0 ist vollständig implementiert, EAI/API zum größten Teil. Die Geschwindigkeit liegt subjektiv etwas höher als beim Cosmoplayer, was möglicherweise auf die insgesamt gröbere Struktur aller Oberflächen zurückzuführen ist. Auch insgesamt wirken VRML 2.0 Welten im Explorer wesentlich grober als beim Cosmoplayer, die Anordnung der Steuerflächen ist aber besser gelungen. Die Lücken in der EAI sind zwar gering, aber leider doch zu bemerken. Außerdem fällt der Explorer durch seine teilweise nicht mehr nachvollziehbaren *ExceptionErrors* unter Java negativ hervor. Für den normalen VRML Betrieb ergeben sich aber keinerlei Ein−schränkungen.

3.3.3 blaxxun Contact 4.0

Dieser Browser ist das Flaggschiff einer umfangreichen Anzahl von VRML Cli−ents welche für die firmeneigenen blaxxun Community Server entwickelt wurden. Neben der vollen Unterstützung des VRML 2.0 Standards enthält der Browser zusätzliche Funktionen zur Benutzerkommunikation innerhalb verteilter virtueller Welten, welche auf den kommerziell genutzten Server der Firma laufen. So gibt es unter anderem ein integriertes Chatapplet auf Java−Basis sowie speziell für den Browser entwickelte animierte Avatare, weiterhin folgen sowohl der Browser als auch der Server weitestgehend den Vorgaben des „*Living World Proposals*, wel−ches als eines der Konzepte für die Realisierung verteilter virtueller Welten im Abschnitt 4.1 näher beschrieben wird . Der Browser ist für Windows und Macin−tosh verfügbar, auf einen Test innerhalb des Projekts wurde allerdings verzichtet da innerhalb der VRML−spezifischen Newsgroups von einigen Schwierigkei−ten bei der Deinstallation des Browsers berichtet wurden, welche Probleme bei der Installation anderer Browser nach sich ziehen sollen. Zu bemerken ist noch das blaxxun und das Web3D Consortium im August 1999 eine kooperative Lizenzvereinbarung (cooperative licensing agreement) abgeschlossen haben, nach welcher der Sourcecode des blaxxun Browsers innerhalb der Arbeitsgruppen des Consortiums frei verfügbar gemacht wird, dies schließt auch eine Modifikation des Codes und freie, nichtkommerzielle Verwendung ein.

3.3.4 Community Place[6]

Dieser, nach Firmenangaben welterste VRML−2.0 Browser wurde von den Soft−ware Labors[7] von Sony entwickelt und ist als eigenständiges Programm für Win−dows erhältlich. Die aktuelle Version 2.0 unterstützt sowohl VRML 1.0 als auch 2.0, Java 1.1/JavaScript, Sound, Direct3D Rendering und Microsofts DirectX Grafik−Beschleuniger Software. Weiterhin besitzt Community Place interne Erweiterungen für verteilte virtuelle Multiuser−Welten in Verbindung mit Sony's Community Place Bureau Server (siehe auch Abschnitt 4.3). Der Benutzer tritt dabei als Avatar auf und kann über die Tastatur mit anderen Benutzern, welche

6 http://www.community−place.com
7 Sony Corporation's Platform Software Development Center (PSDC) in Tokio

sich in der Nähe befinden, kommunizieren. Originell ist dabei die Möglichkeit sich die Texte neben dem IRC−ähnlichen Chatfenster auch als Sprechblase über dem entsprechenden Avatar anzeigen zu lassen.

3.3.5 Liquid Reality

Dieser von der inzwischen bankrotten Firma DimensionX herausgegebene Browser war der erste VRML−Browser welcher vollständig in Java implementiert wurde, und damit weitestgehend plattformunabhängig ist. VRML 2.0 ist fast voll−ständig implementiert worden, herausragend ist aber vor allem die komplette Ein−bindung der VRML−API/EAI, so das die mit Liquid Reality mitgelieferten Java−Klassen lange Zeit als Referenz für die Entwicklung von VRML Applikationen unter Java verwendet wurden [4]. Leider wurde Liquid Reality nach der Über−nahme von DimensionX durch Microsoft nicht mehr weiterentwickelt und spielt heute in der VRML−Welt so gut wie keine Rolle mehr. Ein wesentlicher Grund dafür ist vor allem die Tatsache das Liquid Reality nur unter JDK Versionen bis 1.01 läuft, auf neueren Version ist der Browser nicht lauffähig und somit unbrauchbar.

3.3.6 VRwave[8]

Dieser VRML 2.0 Browser wurde an der Universität Graz am Institute for Infor−mation Processing and Computer Supportet New Media (IICM) entwickelt und unter GPL frei verfügbar. Der Browser ist zu 100% in Java geschrieben und somit quasi plattformunabhängig. Vorcompilierte Binaries sind im Moment für SGI, Sun, Alpha, HPUX und Linux verfügbar. Laut Homepage ist auch eine Version für Windows und Macintosh ab Versionsnummer 1.0 vorgesehen. Im Moment gibt es noch Lücken bei der Implementation des VRML 2.0 Standards und dessen API/EAI, normale virtuelle Szenerien lassen sich aber trotzdem schon sehr gut darstellen. Leider scheint die Weiterentwicklung dieses eigentlich sehr hoff−nungsvollen Projekts in Moment etwas in den Hintergrund getreten zu sein, da das letzte Update auf der VRwave Homepage mittlerweile über 18 Monate zurückliegt.

3.3.7 Worldview 2.0

Der von Platinum Software (www.platinum.com) hergestellte Browser ist als Plugin für Netscape und MS Explorer für Windows 9x und NT verfügbar, eine Version für Macintosh ist (laut Homepage) in Vorbereitung. Neben JavaScript ist auch die Unterstützung von Mirosoft's DirectX API implementiert, ebenso wie die Direct Sound Technology von Microsoft.

3.3.8 FreeWRL[9]

Dieses Gemeinschaftsprojekt verschiedener Linuxprogrammierer (Thomas J. Lukka, John Stewart und andere) zielt auf die Entwicklung einer vollständigen Implementation von VRML 2.0 unter Linux und anderen UNIX Derivaten ab.

8 http://www.iicm.edu/vrwave
9 http://www.crc.ca/FreeWRL/

Obwohl immer noch im Alpha Stadium (Version 0.21 alpha) macht dieser, hauptsächlich unter Perl geschriebene, Browser bereits eine recht guten Eindruck, wenn man einmal von der ziemlich umständlichen und komplizierten Installati−onsprozedur absieht. PROTO und EAI Unterstützung sind mittlerweile fast voll−ständig implementiert, größere Lücken gibt es noch bei spezielleren VRML−Nodes (Texturen, IndexedFaceSets). Eine direkte Unterstützung für VNET (siehe Anhang B) ist für die nächste Zeit angekündigt, damit wäre mit FreeWRL auch wieder ein Browser für dieses System verfügbar der unter UNIX/Linux arbeitet.

3.3.9 Cortona

Dieses Plugin für Netscape und MS Explorer kommt aus der russischen Softwa−reschmiede Parallel Graphics[10]. Neben der vollständigen Implementation von VRML 2.0 inkl. API/EAI ist ein sehr schnelles Software Rendering sowie die direkte Unterstützung von 3D Grafikkarten implementiert worden. Weiterhin ist Cortona der erste VRML Browser welcher speziell für Intels neuesten Pentium III Prozessor entworfen worden ist, aber dank seiner schnellen Grafikroutinen auch auf anderen Prozessoren stabil und schnell läuft. Da dieser Browser erst gegen Ende der Arbeiten an dieser Diplomarbeit erschienen ist konnten keine größeren Tests bezüglich Kompatibilität zu den bestehenden Standards mehr durchgeführt werden. Der erste Eindruck bezüglich Geschwindigkeit, Darstellungsqualität und Steuerbarkeit sind aber durchweg sehr positiv.

10 www.parallelgraphics.com

3.4 Zusammenfassung

Insgesamt kann gesagt werden das die heutige Browsergeneration den aktuellen VRML 2.0 vollständig und abwärtskompatibel darzustellen vermag. VRML 1.0 und 2.0 Browser sind für alle gängigen Betriebsysteme verfügbar, wobei die Anzahl der kommerziellen Applikationen bei weitem überwiegt. Somit sind die meisten Internetseiten, welche VRML Inhalte besitzen, für die meisten Anwender ohne Probleme zu besuchen und auszuführen. Probleme ergeben sich lediglich bei Welten welche neben reinem VRML−Code zusätzliche Funktionen und Applets in Java implementiert haben − hier sind die Plattformen welche von den SGI und Microsoft−Browsern unterstützt werden nach wie vor im Vorteil. Der Cosmo−Player, obwohl bedauernswerterweise von SGI nicht mehr weiterentwickelt, macht seinem Namensgebung als Standard−VRML−Browser alle Ehre. Es bleibt zu hoffen das unter der Schirmherrschaft des Web3D Consortiums die Weiterent−wicklung dieses Browsers weitergeführt wird und auf weitere Plattformen ausge−dehnt wird. Auch die neueste Vereinbarung zwischen blaxxun und dem Consor−tium über eine Freigabe des Quellcodes der blaxxun Browser stellt einen wichti−gen Schritt in der Weiterentwicklung von VRML dar, beinhalten diese Browser doch Funktionen welche in den Entwürfen für die nächste VRML Generation bereits enthalten sind. Über diese und weitere Konzepte zur Vernetzung virtueller Welten wird nun im nächsten Abschnitt berichtet und diskutiert werden.

4 Konzepte verteilter virtueller Welten

Wie bereits im Abschnitt 2.1 beschrieben hatten die bisherigen Entwicklungsstufen von VRML als Hauptziel die Entwicklung eines universellen Standards zur Implementation und Darstellung einzelner oder mehrerer dreidimensionaler Objekte, Szenarien oder Welten im World Wide Web zum Inhalt. Von Anfang an wurde auch eine Vernetzung solcher Objekte und Welten vorgesehen, aufgrund des zu erwartenden Arbeitsaufwandes wurde diese Aufgabe aber als eigenständiges Projekt aus den Entwicklungen zu VRML 1.0 und 2.0 ausgeklammert. Nachdem nun die Stufen 1 (appearance) und 2 (behavior) abgeschlossen wurden (siehe Abschnitt 2.1) wird seit 1998 zur Entwicklung von Stufe 3 (distribution) und damit VRML 200x übergegangen, welche die Vernetzung virtueller 3D−Welten als Hauptinhalt hat. Zurückgegriffen wird dabei hauptsächlich auf einen relativ frühen, aber dennoch bereits sehr umfangreichen Entwurf, welcher als Bestandteil zu VRML 2.0 aber später aus dem Standard herausgenommen wurde. Dieser und andere Konzepte und bereits vorhandene Implementationen sollen im folgenden vorgestellt werden.

4.1 The Living World Proposal

1996 schlossen sich die drei Firmen Black Sun Interactive, Paragraph International und Sony zur *Living World Working Group* innerhalb des VRML Consortiums zusammen[11] und stellten im Rahmen der Ausschreibungen für die Entwicklung der Stufe 3 des VRML Standards den Entwurf eines offenen Standards für verteilte virtuelle Welten unter VRML vor. Genauer genommen handelt es sich dabei um eine Sammlung von Prototypen unter VRML 2.0, welche als Standardschnittstellen zu eingebetteten Java−Applets bzw. Skripts dienen. Der Entwurf selber ist inzwischen in einer etwas abgewandelten Form in den Browsern und Server von blaxxun implementiert und wird warscheinlich im zukünftigen Web3D Standard eingebunden werden. Eine entsprechende Arbeitsgruppe innerhalb des Web3D Consortiums existiert seit 1997 und hat in Zusammenarbeit mit der Avatar−Arbeitsgruppe die Aufgabe, dem Standard entsprechende Prototypen zu implementieren. Der Grundgedanke dabei ist das der Austausch von Daten echt verteilt sein soll. Das heißt, die Clients kommunizieren untereinander ohne Zuhilfenahme eines zentralen Servers. Für die Nutzung soll ein VRML 2.0 kompatibler Browser genügen. Entwurf (Draft) 1 wurde 1997 eingefroren, für Entwurf 2 (siehe [17]) liegen bereits komplette Implementationen vor. Es soll im folgenden auf die Grundeigenschaften und Prinzipien des Standards eingegangen werden, dazu gibt es auf den Seiten des Entwurfes zwei Beispielszenarien welche diese recht anschaulich erläutern und aufzeigen was dieser Standard alles zu leisten vermag. Im folgenden sollen diese Szenarien näher betrachtet werden, wobei teilweise auf die Kommentare in [17] zurückgegriffen wurde:

11 Parallel zur universal avatar working group die den Entwurf und die Spezifikation von universell einsetzbaren Avataren unter VRML zur Aufgabe hat.

Szenario 1− "Ein Abend zu Hause", zeigt den Benutzer Art welcher sich in seinem virtuellen Wohnzimmer befindet. An den Wänden hängen Bilder welche einmal wöchentlich von einer ebenfalls virtuellen Galerie ausgetauscht werden. Irgendwann an diesem Abend kommen seine Freunde Betty und Shuck zu Besuch, Betty bringt zusätzlich ihren Kanarienvogel mit welcher im Zimmer umherfliegt und dabei Federn verliert. Die drei Benutzer starten eine angeregte Diskussion, in dessen Verlauf Art eine Zeichnung auf ein Witheboard zeichnet, welche von Betty und Shuck ergänzt wird. Anschließend spielen die drei noch eine virtuelle Variante des klassischen Brettspieles Monopoly welches als beson− deres Merkmal die Möglichkeit bietet die Avatare der Benutzer als Spielsteine so weit schrumpfen zu lassen bis die Häuser und Hotels in Lebensgröße erscheinen.
Zwei Tage nach dem Spiel bemerkt Art das Bettys Kanarienvogel ihm ein Geschenk hinterlassen hat − ein blaues Ei mit gelben Flecken. Als er dieses anklickt fängt es an zu zittern und bricht nach kurzer Zeit auseinander, worauf sich eine Schar von Kriegern mit Sandalen und geschmückten Helmen daraus ergießt welche Drohungen in Griechisch ausstoßen und anfangen mit ihren Strei− täxten die Bilder von der Wand schlagen.

Diese Szenerie hat in ihrem Verlauf verschiedene Probleme bezüglich Dateninte− grität und Zugriffsrechten aufgeworfen. So ist die Szene am Anfang, wo Art noch alleine im Zimmer sitzt, zwar noch *single−user* , der Raum selber aber stellt bereits eine verteilte Umgebung dar da die Bilder an den Wänden von einer externen Firma bereitgestellt werden. Nachdem Art Besuch bekommen hat ist die Szene *multi−user* geworden, was weitere Fragen aufwirft: Wie sehen sich die Personen untereinander? (Avatare). Wie kann Art auf seinem Computer hören das Betty an die Tür geklopft hat bevor sie das Zimmer betreten hat? Wie ist es über− haupt möglich Avatare und Roboter (Bettys Kanarienvogel ist kein echter Avatar sondern nur ein animiertes Objekt) auf andere Dinge zugreifen zu lassen (White− board, Ei, die Krieger welche das Wohnzimmer zerstören)? Besonders das letztere Beispiel eines „trojanischen Pferds" zeigt ein Problem auf welche bei der Ent− wicklung eines solchen Entwurfes unbedingt berücksichtigt werden muß: die Bereitstellung verläßlicher Schutzmechanismen wie bei jeder anderen Netzwer− kanwendung.

Szenario 2: Eine virtuelle Verkaufsaustellung

Um die Attraktivität ihrer populären Verkaufsaustellungen noch zu steigern, beschließt SoftBank ihre Frühjahrsmesse um einen virtuellen Führer zu ergänzen.

Austellungen und Messen stellen eine komplexe Gemeinschaft ineinander ver− flochtener Interessen dar: Verkäufer, Besucher, Produzenten, Geschäftsleute.

Die Verkaufsstände und deren Personal (Robots oder Avatare) werden ebenso wie die herumlaufenden "Advertars" (eine Art wandelnde Litfaßsäule mit den Namen von Sponsoren) extern erzeugt und gesteuert.

Dieses Konzept zeigt einen weiten Bereich von Problemen auf welche in den bis− herigen VRML Standards nicht vorgekommen sind, da sich hier nur mit dem Aussehen und den Bewegungsmöglichkeiten von Objekten beschäftigt wurde.

Gemeinschaften, wie eben auch diese relativ kurzlebigen Messen, benötigen vor allen eines: umfassende Möglichkeiten zu Kommunikation untereinander als auch Interaktion mit den entsprechenden Ergebnissen.

Alle Besucher bekommen Ausweise welche von den Ständen gelesen werden können und ihrerseits deren Daten einer austellungsweiten Datenbank zur Verfügung stellen.

Hier stellen sich sowohl Probleme bezüglich der Sicherheit/Vertrauenswürdigkeit von virtuellen Ausweisen als auch die Frage, bei wem die Entscheidung liegt welche Informationen an die Datenbank weitergegeben werden dürfen und welche nicht.

Die Ausstellung beinhaltet auch eine Konferenz(saal) − in welchem Vorführungen stattfinden, Teilnehmer Fragen stellen bzw. sich an Diskussionsrunden und Umfragen beteiligen.

Frage: Was ist das VRML−Äquivalent eines Hörsaals? Wie muß ein Vortragender in einer Präsentation auftreten und kommunizieren wenn die Große der Zuhörerschaft die Möglichkeiten des Renderings jedes einzelnen Zuschauers auf den meisten Desktop−Computern übersteigt?

Die virtuelle Ausstellung wurde zwei Wochen vor der realen Ausstellung, auf welcher die Besucher eine CD mit neuerem Material erhalten haben, bereitgestellt.

Die neue Welt (auf der CD) muß sich nahtlos in die bereits vorhandene Welt einfügen können. Dies ist weniger ein neues Problem in der Software Welt, umso mehr bekommt es aber einen neuen Charakter wenn sich die Applikationen während der Benutzung dynamisch entwickeln sollen.

4.1.1 Eigenschaften

In [4] und [13] werden die Anforderungen an den Standard wie folgt zusammengefaßt:

- Einfügen und Löschen von Objekten , z.B. für Avatare
- Zusammenführung von Audioströmen
- Verfolgung und Weiterleitung des Standortes und Zustandes von Objekten
- Möglichkeit von Bewegung von Objekten in Echtzeit durch den Benutzer
- Objekte können persistent werden
- Schutz von Szenen vor Zerstörung
- Einführung von Nutzerrechten und Eigentumsverhältnissen
- Unterstützung persistenter Rollen
- Dynamisches Linken von Objekten zu externen Daten und Funktionen
- Unterstützung des Datenaustausches zwischen Objekten

4.1.2 Das Living World Modell

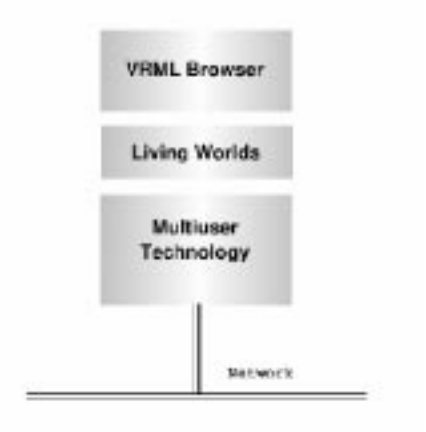

Abbildung 3: Das Living World Schichtenmodell

Anhand eines Schichtenmodells kann man Living Worlds als dünne Schicht zwischen den wesentlich größeren Schichten VRML Browser und *MUtech* einordnen, wobei der Entwurf selbst nichts über das Design und Technik der *MUtech* Schicht aussagt. Dies bleibt dem jeweiligen Entwickler überlassen. Andere Konzepte wie Open Community und Cyber Sockets dagegen beschreiben den Aufbau und Implementation einer *MUtech* API in ihren Spezifikationen.

Die Komponente welche die serverlose Kommunikation zwischen den Clients ermöglicht ist der sogenannte *MUtech (*Abkürzung für Multi−User−Technologie*)*, welcher auf jedem Client vorhanden ist. Dabei handelt es sich nicht um eine ausführbare Datei sondern um einen Prototypen gemäß des VRML 2.0 Standards. Dieser wird bei jedem Aufruf der virtuellen Welt in den Browser geladen und kommuniziert dann über entsprechende Java−Skripte mit dem Browser und den anderen angeschlossenen Clients.

4.1.3 Nodes

Im Living Worlds Proposal werden im wesentlichen zwei neue Nodes eingeführt: *SharedObject* und *Zone.* Dabei handelt es sich um Implementationen von VRML 2.0 Prototypen so das die unter Living World laufenden Welten von jedem VRML 2.0 kompatiblen Browser dargestellt werden können ohne das weitere Plugins oder externe Applikationen verwendet werden müssen.

SharedObject: Dieser Knoten stellt eine Kapselung für verteilte Objekte dar um sie von rein lokalen Objekten zu unterscheiden. Jedes *SharedObject* besteht aus zwei Bestandteilen − dem gleichnamigen *SharedObject* Teil welcher allgemeine und von außen zugängliche Informationen über den Status eines verteilten Objekt enthält, wie zum Beispiel Standort, Persistenz, Eigentümer usw. Die für die Verteilung eines Objektes wichtigen Schnittstellen befinden sich im zweiten Teil, dem *PrivatSharedObject*. Auf diesen Teil kann nur die *MUtech* des Eigentümers zugreifen. Hier macht man sich die Sicherheitsbestimmungen der VRML Spezifikation zu Nutze nach denen es nur möglich ist auf ein Objekt manipulierend

zuzugreifen wenn man mittels eines Zeigers direkt auf den Knoten gelangen kann. Da das eigentliche VRML Objekt als URL innerhalb vom *PrivatSharedObject* liegt kann nur das berechtigte *MUtech* darauf zugreifen, auch wenn sich die Datei selbst auf einem entfernten Rechner befindet.

Zone: Eine Zone ist ein sowohl räumlich als auch konzeptionell zusammenhängender Teil einer Szene (zum Beispiel das Monopolyspiel im Szenario 1), somit also auch eine Sammlung von *SharedObject* Knoten. Der Grund für solch eine Gruppierung liegt in der Arbeitsweise der *MUtech* begründet welche jeweils einen konsistenten Arbeitsbereich benötigt. Das obige Beispielszenario hat gezeigt das durchaus mehrere unabhängige, von verschiedenen Entwicklern erzeugte *MUtech*s in einer Szene vorkommen können, hierbei muß gewährleistet sein des jede *MUtech* einen genau abgegrenzten Zuständigkeitsbereich besitzt. Erschwerend kommt noch hinzu das die *SharedObject*s flexibel gehandhabt werden müssen, das heißt ein Objekt kann eine Zone und somit auch die *MUtech* wechseln. Auch die *Zone* besitzt einen als privat deklarierten Teil, die *PrivatZone*. Analog zum *PrivatSharedObject* sind auch hier alle Schnittstellen zu finden welche zur Verteilung der Szene notwendig sind und auf die nur die *MUtech* Zugriff hat.

4.1.4 Kommunikation zwischen den Living World Clients

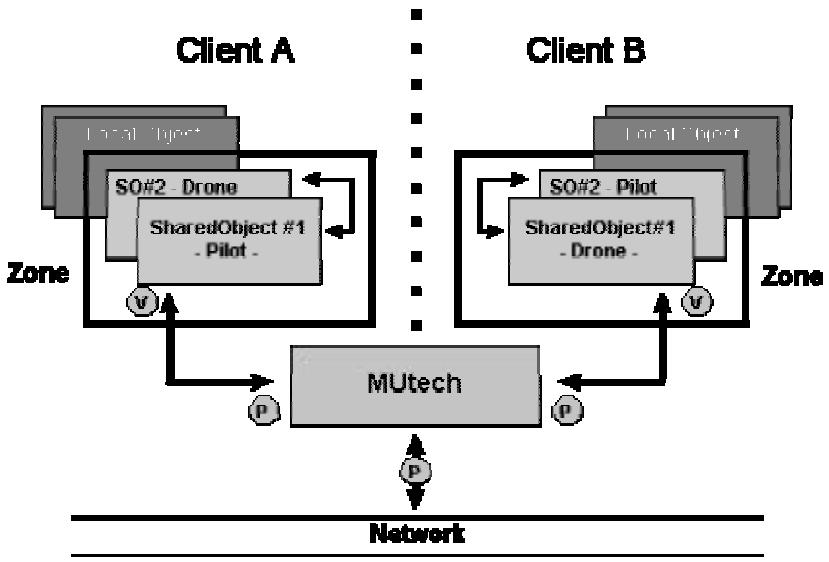

Abbildung 4: Kommunikation zweier Clients über MUtech

Abbildung 4 zeigt wie zwei Clients über das Netzwerk miteinander kommunizieren. Dabei handelt es sich nicht um eine Verbindung welche über einen gemeinsamen Server läuft, vielmehr benutzen beide Clients dieselbe Komponente einer *MUtech* welche auf beiden Rechnern vorhanden ist. Ein Unterstützung der Kommunikation über einen zentralen Server ist unter Living Worlds zwar möglich, prinzipiell aber nicht vorgesehen. Die einzelnen *MUtechs* tauschen untereinander Daten aus. Die Abbildung zeigt also nicht eine einzelne *MUtech* sondern das die Clients jeweils auf die selbe Instanz ihrer eigenen *MUtech* zugreifen. Dies wirft das Problem auf das auf jedem Client alle Instanzen der jeweils aktuellen Szene

vorhanden sein müssen, jede Instanz kann aber nur von einem Ort kontrolliert werden kann. Dies wird dadurch gelöst das es zu jedem Objekt jederzeit genau eine Instanz auf einem der beteiligten Rechner gibt welche die Kontrolle über das Objekt hat. Diese Instanz wird als *Pilot* bezeichnet, alle anderen, passiven Instanzen als *Drone*.
Ein *Pilot* teilt Veränderungen an seinem Objekt den *Dronen* mit welche diese Veränderungen dann replizieren. Die eigentliche Zuordnung von *Pilot* und *Drone* an einen bestimmten Rechner im Netz ist in *Living Worlds* aber nicht statisch, vielmehr kann diese innerhalb einer Szene wechseln. So ist im Falle des Beispielszenarios die Kontrolle des Würfels des Monopolyspieles (und somit der *Pilot* des Objekts Würfel) jeweils bei der Person welche den Würfel hält. Diese Kontrolle ändert sich im Laufe des Spieles regelmäßig. Anders bei Avataren, hier liegt eindeutigerweise die Kontrolle nur beim Benutzer den der Avatar in der Szene repräsentiert. In jedem Falle gilt das in einer Zone zu jedem Zeitpunkt nur genau eine Instanz Pilot sein darf.

4.1.5 MUtech

MUtech steht, wie bereits erwähnt, für Multi−User−Technology und bildet das Kernstück des Living World Standards. Die *MUtech* ist verantwortlich für die Kommunikation zwischen den Clients, der Verteilung von Objektzuständen sowie zum Synchronisieren von Szenen. Außerdem enthält sie sicherheitsrelevante Schnittstellen. Entgegen den anderen Komponenten des *Living Worlds Proposal* ist die *MUtech* als proprietäre Einheit geplant worden, welche gemäß den Anforderungen der entsprechenden verteilten Welt implementiert wird. Gefordert wird nur das alle Clients einer virtuellen Welt die gleiche MUtech benutzen müssen. Diese scheinbare Einschränkung der Flexibilität wird dadurch aufgehoben das die *MUtech* als VRML 2.0 kompatibler Prototyp implementiert wird, das heißt beim laden der Welt auf den Rechner wird gleichzeitig auch der *MUtech* Prototyp geladen und kann anschließend die ihm zugedachten Kommunikationsaufgaben ausführen. Bei der Implementation sind lediglich einige Grundfunktionen vorgegeben, wie zum Beispiel das Weiterleiten von Ereignissen an den *Pilot* oder die Replikation durch *Drone*.

4.1.6 Status der Implementation

Die letzte Eintragung der Living World Arbeitsgruppe auf der Homepage des Web3D Consortiums stammt vom Dezember 1998. Die Spezifikation des Standards ist inzwischen abgeschlossen worden. Über den Stand der Implementierung von verteilten Welten mittels Living World Clients ist aber bis jetzt nichts bekannt, wohl auch deshalb weil sich das Hauptaugenmerk im letzten Jahr verstärkt auf eine Kooperation mit *blaxxun* gerichtet hat, was letztendlich zu einem internen *Open−Licence Agreement* für die Browser von *blaxxun* innerhalb des Web3D Consortiums geführt hat. *Blaxxun*, als Mitinitiator des Living World Standards, stellt damit seine fortgeschrittene Technologie der VRML Gemeinde zur Verfügung, so bleibt zu hoffen das Living Worlds als integraler Bestandteil des künftigen VRML 200x Standards bald vollständig implementiert sein wird.

4.2 Open Community

4.2.1 Überblick

Open Community wurde/wird von Mitsubishi Electric Information Technology Center America entwickelt. Im Gegensatz zu *Living Worlds* handelt es sich hier−bei um eine Software Bibliothek welche nicht unmittelbar zur Unterstützung von VRML entwickelt wurde, statt dessen als verteilte Umgebung zur Verwaltung von Echtzeit Multi−User Systemen. Allerdings wird in den Dokumentationen aus−führlich auf den Einsatz unter VRML 2.0 eingegangen. Die Software selber ist unter Java und C verfügbar. Anwendungen die mit der *Open Community Library* verbunden sind sehen diesen Dienst als eine spezielle Form von Shared Memory in der Form einer objektorientierten Datenbank (in den Dokumenten als World Model bezeichnet) welche in jeder Applikation reproduziert wird (in Wirklichkeit wird nur ein Teil der Datenbank wirklich reproduziert, für den Anwender sieht es aber stets so aus als ob die gesamte Datenbank vorliegt). Alle Objekte im World Model sind Klassen von Objekten aus der Library bzw. von Usern abgeleitete Objekte der Datenbank.

4.2.2 Eigenschaften

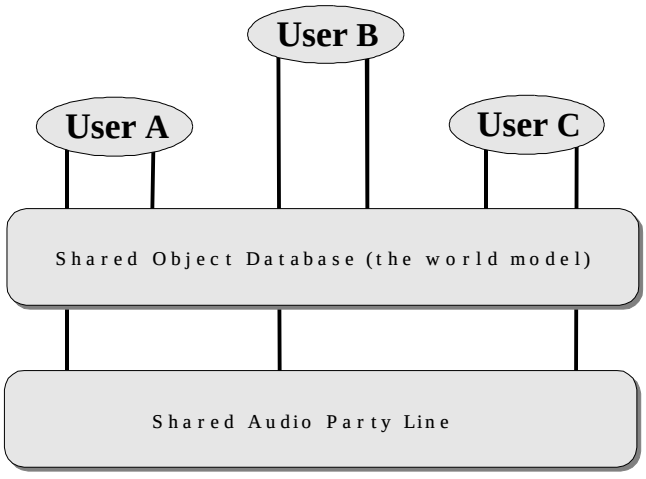

Abbildung 5: Prinzip von Open Community

Wie Abbildung 5 zeigt sind alle User Prozesse (A, B, C) auf die Datenbank sowohl lesend als auch schreibend zugreifen können, ebenso sprechend und hörend auf der *Shared Audio Party Line*. Dieses einfache Model erzeugt nach [14] die folgenden Anforderungen an das System:

- Open Community Datenobjekte erscheinen als verteilte Objekte
- Jeder kann Datenobjekte erzeugen und diese werden ebenso für jeden sichtbar
- Nur der Eigentümer eines Objekts kann dessen Eigenschaften verändern bzw. modifizieren (wobei Eigentumsrechte übertragen werden können)
- Alle diese Eigenschaften sollten auch in einem weitverzweigten Netzwerk bestehen bleiben.

Die selben Anforderungen gelten natürlich auch für die Audio−Schicht:

- Jeder kann sprechen
- Alles was in einer Welt gesagt wird kann auch von den anderen Benutzern gehört werden. In der Praxis bedeutet das daß natürlich nur die Personen ein Gespräch hören welche nah genug an dessen Quelle stehen.
- Alle Eigenschaften bleiben auch in einem großen Netzwerk bestehen.

4.2.3 Das Open Community World Model

Unter Open Community gibt es drei Arten von Objekten:

- Objekte die relativ unbeweglich sind wie z.B. Gebäude oder Flächen
- Objekte die sich bewegen (Menschen und Fahrzeuge)
- Objekte nur nur für eine sehr kurze Zeit existieren und dann wieder ver−schwinden (Audioströme)

Es gibt noch weitere spezielle Objekte welche in den Dokumenten als *Locales*, *Beacons,* und *Actors* beschrieben werden und auf die hier nicht weiter eingegan−gen wird. Die Open Community Datenbank selber enthält dabei weder das gesamte Objekt noch Teile davon, vielmehr wird lediglich eine URL übertragen unter der dann das Objekt in einer entsprechenden Beschreibungssprache vor−liegt[12]. Der Browser des Users kann dann das Objekt unabhängig von Open Com−munity zeichnen und darstellen. Der Standort eines Objekts kann entweder im klassischen 4x4 Matrixsystem dargestellt werden oder unter dem 17 elementigen System, welches VRML 2.0 verwendet[13] (und voll kompatibel zu diesem ist).
Zusätzlich zu diesen Definitionen des Standortes eines Objekts fügt Open Com−munity noch zwei Ergänzungen hinzu:

- Alle Objekte können Unterobjekte haben, so das wenn sich ein Objekt

12 Open Community ist ein reines Datenbanksystem welches selber keinerlei Standards zur Darstellung von 3D Welten implementiert hat.

13 Wobei neben den normalen XYZ Koordinaten auch Winkel für Neigung, Blickrichtung, Dre−hung usw. angegeben werden

bewegt alle Unterobjekte ebenfalls mitbewegen (also eine Implementation die mit den 'Grouping Nodes' unter VRML 2.0 vergleichbar ist).

- Anstelle eines einzigen allumfassenden Koordinatensystems für alle Objekte erlaubt Open Community die Nutzung unendlich vieler unabhängiger Koordinatensysteme. Diese Koordinatensysteme unterteilen den virtuellen Raum in Regionen welche als Locale bezeichnet werden. Auch hier ein guter Vergleich zu VRML wo mittels Transform−Knoten Objekte in verschiedene Koordinatensysteme transformiert werden können.

4.3 Community Place Bureau

Dieser von Sony entwickelte kommerzielle Multi−User Server arbeitet direkt mit dem von der gleichen Firma bereitgestellten VRML−Browser Community Place (siehe Abschnitt 3.3.4) zusammen und ermöglicht durch interne Erweiterungen die Vernetzung der Browser zu einer Multi−User−Welt. Die User treten dabei in Form von Avataren auf, die Kommunikation erfolgt über ein textbasiertes Chat−system, welches die eingegebenen Texte wahlweise entweder in einem IRC−ähnlichen Textfenster erscheinen läßt oder als Sprechblase über dem entsprechenden Avatar.

4.4 blaxxsun[14]

blaxxun ist eines von Black Sun Interactive entwickeltes virtuelles Chatsystem welches auf VRML aufsetzt und im Moment das verbreitetste kommerzielle 3D System im WWW. Zum Einsatz kommt ein von derselben Firma entwickelter VRML Server und eine Anzahl von Browsern (siehe Abschnitt 3.3.3), die sowohl den VRML 2.0 Standard vollständig und die EAI Spezifikation weitestgehend vollständig implementiert haben als auch einige zusätzliche (firmeneigene)Erweiterungen unterstützen. Aus diesem Grund können die von Blaxxun bereitgestellten virtuellen Welten nur mit deren eigenen Browsern besucht werden. Der Browser selbst ist als kostenloses Plugin für Netscape Navigator bzw. MS Internet Explorer unter Windows und Mac erhältlich. Black Sun selbst war, in Zusammenarbeit mit Paragraph International und Sony, maßgeblich an der Entwicklung des *Living World Proposal* beteiligt welcher bereits im Abschnitt 4.1 erläutert wurde.

4.5 VNET

VNET ist ein von Stephen White und Jeff Sonstein geschriebenes Virtual Reality System welches nur die offenen Standards von VRML 2.0 und Java verwendet. Es ist frei verfügbar und wird unter der GPL entwickelt und vertrieben. Zum Betrieb ist lediglich ein zu VRML 2.0 kompatibler Browser notwendig. Das Java Chatapplet wird je nach Typ des Browsers entweder als eingebettetes Applet oder in einem externen Fenster angezeigt. VNET benötigt eine weitestgehend vollständige Implementierung des VRML 2.0 Standards und der EAI bei den verwendeten Browser, so das im Moment der Einsatz unter Unix/Linux (außer SGI) noch problematisch ist. Allerdings beschäftigt sich im Moment mindestens ein VRML−

14 http://www.blaxxun.com

Projekt[15] mit der Implementierung der kompletten EAI Spezifikation unter Linux, speziell auch im Hinblick auf VNET. Die technische Spezifikation von VNET wird später gesondert im Abschnitt 11 besprochen.

4.6 Deep Matrix[16]

Das von der Firma Geometrek entwickelte Multi−User System ist ähnlich wie VNet vollständig unter Java und VRML 2.0 implementiert worden und wird als Open Source frei vertrieben. Es ermöglicht die Vernetzung einer oder mehrerer virtuellen Welten in die der Benutzer nach Wunsch wechseln kann. Zusätzlich unterstützt Deep Matrix auch die Vernetzung von Objekten in der virtuellen Welt, allerdings nicht in dem Maße wie im vorliegenden Projekt. Zum Beispiel sieht ein Benutzer der sich zu einem späteren Zeitpunkt einloggt nicht die Ereignisse wel−che bisher eingetreten sind. Die Kommunikation der User erfolgt auch hier über ein Chat Applet in Java. Im Gegensatz zu VNET unterstützt Deep Matrix auch ansatzweise die Vernetzung von Objekten, wodurch dann Aktionen an diesen in allen angeschlossenen Welten sichtbar werden. Allerdings fehlt z.Zt. noch eine Art Ereignisspeicher, d.h. bereits ausgeführte Aktionen können durch später ein−loggende Clients nicht mehr nachvollzogen werden.

4.7 Zusammenfassung

Obwohl noch nicht im Standard enthalten funktioniert die Vernetzung von VRML Welten in der Praxis bereits recht gut. Dabei werden unterschiedliche Wege gegangen welche teilweise auch kritisch betrachtet werden sollten. So willkom−men die integrierten Funktionen der blaxxun und Sony Browsers auch sein mögen, als fest integrierter Bestandteil des Programms und damit dessen Codes stellen sie eine nicht konforme Erweiterung bestehender Standards dar und die−nen daher hauptsächlich den kommerziellen Interessen der entsprechenden Fir−men. Ähnliches ist ja durch die Entwicklung von HTML bereits bekannt. Auf der anderen Seite stehen Systeme wie VNET und Matrix welche die fehlenden Funk−tionen des aktuellen VRML Standards durch externe Java−Applets ersetzten und damit kompatible und konforme Endlösungen geschaffen haben. Im folgenden Abschnitt soll nun dieser Bereich des externen Zugriffs auf VRML Szenarien aufgezeigt und erläutert werden.

15 Free−WRL, www.freewrl.org
16 http://www.geometrek.com/products/deepmatrix.html

5 Das VRML EAI

Für die Kommunikation zwischen der virtuellen Welt und dessen Umgebung wird ein Interface benötigt welches die nötigen Standardschnittstellen und −methoden bereitstellt. Im Falle von VRML ist dies das External Authoring Interface (kurz EAI), welches ein Paket von Funktionen für den VRML−Browser bereitstellt welche es ermöglichen, direkt auf die Objekte der virtuellen Welt zuzugreifen bzw. selbst neue Objekte einzufügen und zu entfernen. Prinzipiell kann dieser Zugriff durch jede höhere Programmiersprache durchgeführt werden, Voraussetzung ist lediglich das Vorhandensein von entsprechenden VRML Datentypen und −klassen in der jeweiligen Sprache[17] . Weitere Voraussetzung ist die Unterstützung des EAI durch den VRML Browser selbst, dabei wird zusätzlich noch zwischen teilweiser und vollständiger Implementation unterschieden:

Teilweise Implementation
Von teilweiser Implementation spricht man wenn nur Teile der in der EAI Spezifikation von VRML 2.0 aufgeführten Klassen und Schnittstellen implementiert wurden und somit nicht die volle Bandbreite der möglichen Funktionen nutzbar ist. Dies gilt prinzipiell auch für Browser welche die VRML 2.0 Spezifikation selbst nicht vollständig implementiert haben.

Vollständige Implementation
Eine vollständige Implementation der VRML EAI setzt zunächst auch die vollständige Unterstützung des VRML 2.0 Standards inklusive aller Schnittstellen, Syntaxregeln und Grammatiken der Sprache voraus. Weiterhin müssen alle Klassen und Befehle gemäß der VRML−EAI Spezifikation implementiert sein. Browser die die EAI vollständig implementiert haben sind zur Zeit SGI´s Cosmo Player 2.1, Microsofts VRML−Plugin für Internet Explorer 4.x (mit leichten Einschränkungen in der Kompatibilität, siehe auch Abschnitt 6.0.2) sowie der nicht mehr weiterentwickelte Liquid Reality Browser (DimensionX/Microsoft). Besonders unter Linux/Unix gibt es im Moment eine ganze Reihe von VRML−Browsern die das EAI zumindestens teilweise implementiert haben (FreeWRL).

17 Im Moment wird VRML von Java, C/C++ sowie Visual Basic unterstützt

5.1 Aufbau eines VRML Browsers

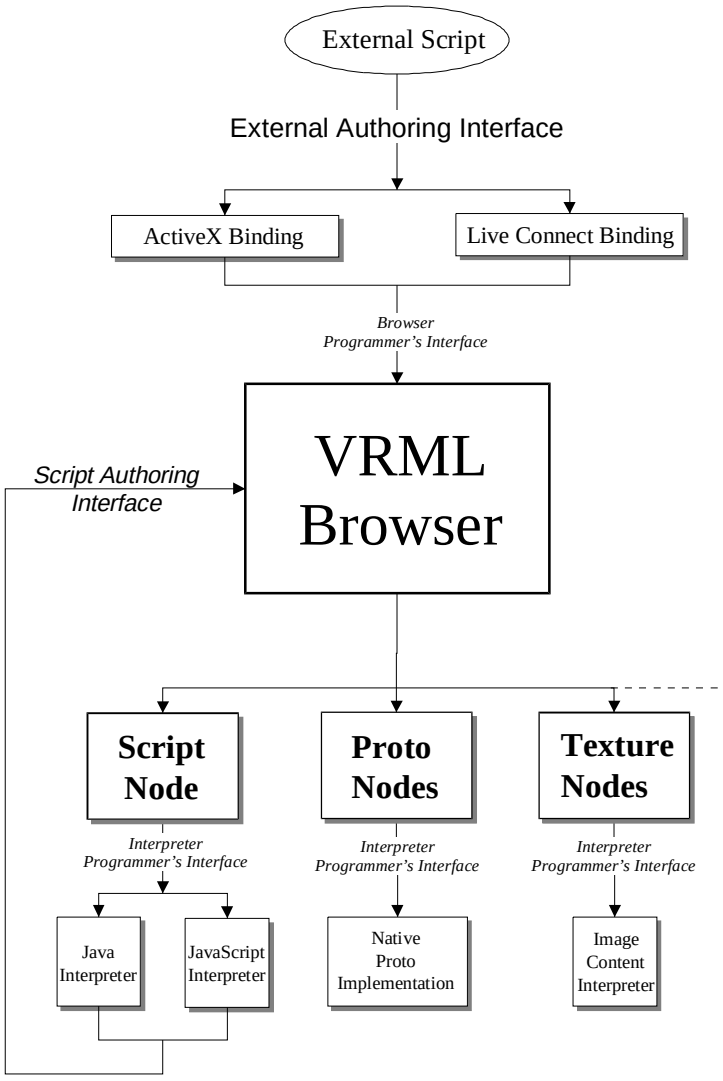

Abbildung 6: Prinzipieller Aufbau eines VRML−Browsers mit EAI und SAI

Das Diagramm in Abbildung 6 (aus http://www.marrin.com/vrml/Interface.html) zeigt einen VRML 2.0 Browser mit EAI und SAI (Script Authoring Interface) Unterstützung mit einigen Interfaces zu Standardanwendungen (Skripts, Prototy−pen, Bilder). Das SAI verwendet oftmals den im HTML Browser vorhandenen JavaScript Interpreter (Cosmo, MS−Explorer) und hat daher lediglich die erfor−derlichen VRML−Klassen bereitzustellen. Dies gilt prinzipiell auch für das EAI und den Java Interpreter, beide benutzen die gleichen Schnittstellen. Es gibt aber wichtige Unterschiede in Hinblick auf die Arbeitsweise dieser Schnittstellen, die wichtigsten

Dabei sind:

Laufzeitumgebung: Skripte, welche durch das SAI ausgeführt werden, laufen stets innerhalb des VRML−Browsers ab, und es ist nicht möglich mit externem Java−Code zu kommunizieren, mit Ausnahme von Applets. Andererseits läßt sich das EAI prinzipiell nur über Applets ansprechen, es kann nicht als eigenständiges Programm existieren [4] [15].

Pakete: Beide Systeme verwenden unterschiedliche Teile des VRML−Gesamtpa−kets. Während Skripte die Pakete vrml, vrml.node und vrml.field benutzen ver−wenden Applets die Pakete der der vrml.external Hierarchie.

Ereignisse: Während in Skripten jedes eintreffende Ereignis durch den entspre−chenden Datentyp definiert werden muß bei Verwendung des EAI jede Variable welche externe Ereignisse empfangen soll vorher als solche registriert werden (Callback).

Initialisierung: Skripte und externe Applets besitzen beide eine Implementation des Browser Interfaces. Allerdings benötigt ein Applet, im Gegensatz zum Skript, weitere Informationen bezüglich der Umgebung in der es sich befindet. Es wer−den daher wesentlich mehr Schritte benötigt um ein Applet in Hinblick auf das EAI zu initialisieren als dies bei einem Skript der Fall ist (siehe Abschnitt 5.3).

5.2 VRML−Datentypen und −Klassen

Dieser Abschnitt beschreibt die im VRML 2.0 Standard definierten Datentypen und −Klassen welche sowohl intern als JavaScript Routinen (VRMLScript in VRML 1.0) als auch extern von Hochsprachen wie Java oder C/C++ bereitgestellt werden müssen.

Hinweis: Meistens sind diese Klassen in den jeweiligen Entwicklungsumgebun−gen nicht vorhanden und müssen zusätzlich besorgt werden. Zum Beispiel enthält das JDK (Java Development Kit) von Sun keinerlei VRML−Unterstützung, die benötigten Klassen werden aber beispielsweise bei den VRML−Browser von Sili−con Graphics und Microsoft mitgeliefert.

5.2.1 Datentypen

Ähnlich wie in höheren Programmiersprachen können die meisten der in VRML verwendeten Datentypen sowohl einzeln als auch als Array deklariert werden. Dabei ist allerdings zu beachten das einige Datentypen von vornherein mehrere Übergabeparameter besitzen (z.B. SFVect3f mit drei Parametern, SFRotation mit vier usw.), daher ist die Bezeichnung einzelnes Feld (single Field) hier zutreffen−der.

Die VRML 2.0 Spezifikation definiert die folgenden Datentypen:

Single Field	**Multiple Field**
SFBool	
SFInt32	MFInt32
SFFloat	MFFloat
SFString	MFString
SFVect2f	MFVector2f
SFVect3f	MFVector3f
SFNode	MFNode
SFTime	MFTime
SFImage	
SFRotation	MFRotation
	MFColor

SFBool ist ein einzelner Boolean Wert. SFBools werden als TRUE oder FALSE geschrieben:

```
fooBool FALSE
```

SFInt32 ist ein 32 Bit Integer Wert, der sowohl in dezimaler als auch hexadezimaler Schreibweise angegeben werden kann:

```
fooInt32 [ 17, -0xE20, -518820 ]
```

SFFloat ist ein *single-precision* Fließkommawert, welcher im ISO-C-Format angegeben wird:

```
fooFloat [ 3.1415926, 12.5e-3, .0001 ]
```
[18]

SFString ist ein im *UTF-8 universal character set* formatierter String welcher in Anführungszeichen (" ") gesetzt wird. Es können alle Zeichen inkl. Zeilenumbrüche verwendet werden, ein Anführungszeichen kann durch einen davorgestellten Backslash (\) dargestellt werden[19]:

```
fooString ["Eins, Zwei, Drei", "Er sagte, \"Immel
        war es!\""]
```

SFVector2f stellt einen zweidimensionalen Vektor dar welcher als ein Paar von

18 Zur besseren Übersicht sind hier alle Schreibweisen in einem einzigen MFFloat Feld angegeben worden.

19 Ein Backslash selber wird dann wiederum durch einen zweiten vorrangestellten Backslash dargestellt

ISO−C−formatierten Fließkommawerten dargestellt wird:

```
fooVec2f [ 42 666]
```

SFVector3f stellt einen zweidimensionalen Vektor dar welcher als ein Paar von ISO−C−formatierten Fließkommawerten dargestellt wird:

```
fooVec3f [ 1 42 666 ]
```

SFNode spezifiziert einen VRML Knoten (Node). Die Beschreibung hat in der von der VRML Spezifikation vorgegebenen Syntax zu erfolgen.

SFTime ist ein *double precision* Fließkomma Wert welcher eine Zeitangabe repräsentiert. Normalerweise wird dieser Wert als Anzahl der Sekunden angegeben welche seit einen bestimmten Zeitraum vergangen sind. Die meisten Knoten verwenden dafür den 1. Januar 1970 um 0.00.00 Uhr UTC.

SFImage spezifiziert ein einzelnen zweidimensionales, unkomprimiertes Farb− oder Graustufen−Image. SFImage Felder enthalten zwei Integer Werte für die Breite und Höhe, einem Integer (1−4) für die Anzahl der Komponenten pro Bildpunkt:

1 Byte: Grauwert
2 Bytes: Grauwert, Opazität
3 Bytes: Rot−, Grün−, Blauwert
4 Bytes: Rot−,Grün−,Blauwert, Opazität

gefolgt von Breite−Höhe Hexadezimalzahlen welche durch Leerzeichen getrennt sind. Jede hexadezimale Zahl repräsentiert ein einzelnes Pixel des Bildes:

```
fooImage 2 4 3 0xFF0000 0xFF00 0 0 0 0 0xFFFFFF 0xFFFF00
# red    green  black.. white    yellow
```

SFRotation spezifiziert eine definierte Drehung, wobei die ersten drei (Fließkomma−) Werte den Vektor der Drehachse angeben, der vierte Fließkommawert gibt den Drehwinkel im Bogenmaß an:

```
fooRot 0.0 1.0 0.0 3.14159265
#rotation um 180 Grad um die y−Achse
```

5.2.2 Feldtypen

Variablen in Nodes können über die EAI Parameter von externen Applikationen entgegennehmen bzw. abgeben, hierzu stehen die Feldtypen: **EventIn** (Übergabe von Parametern an den Node), **EventOut** (Abruf vom Parametern aus dem Node) sowie **exposedField** (Kombination von EventIn/Out) zur Verfügung.

Wichtig: Der Zugriff auf die Datenfelder ist auf die Nodes und Prototypen im Basisfile (der obersten Datei einer VRML Hierarchie) beschränkt, es gibt daher

keine Möglichkeit die vernetzten Objekte wegen der besseren Übersichtlichkeit in eingebettete Dokumente (genannt *Inline*−Dateien) einzutragen.

5.3 Zugriff auf die EAI in Java

Wie schon im Abschnitt 5.1 angedeutet muß einem Applet vor dem eigentlichen Zugriff über das EAI mitgeteilt werden in welcher Umgebung es sich befindet. Durch den Aufruf von:

```
public class Browser
```

werden dem Applet die dafür notwendigen Laufzeitparameter mitgeteilt. Dabei stehen zusätzlich die Methoden:

```
public String   getName();
public String   getVersion();
```

zur Verfügung. Anschließend kann durch die mittels der Klasse *Browser* initiali−sierten Variablen auf die einzelnen Nodes zugegriffen werden:

```
Browser browser = (Browser)
      vrml.external.Browser.getBrowser(this);
Node klicker = browser.getNode("Klicker");
```

Auf den durch die Variable *node* repräsentierte Node kann nun auf die entspre−chenden Datenfelder des äquivalenten VRML Datentyps zugegriffen werden:

```
EventOutSFBool active = (EventOutSFBool)
              klicker.getEventOut("isActive");
EventInSFBool status = (EventInSFBool)
              klicker.getEventIn("value_changed");
```

Senden von Daten an einen Node

Die Übergabe von Parametern an einen als *EventIn* deklariertes Feld in einem Node erfolgt durch die Methode *setValue(value)*:

```
currentStatus=TRUE;
status.setValue(currentStatus());
```

Zugriff auf einen EventOut

Um Daten von als *EventIn* deklarierten Feldern lesen zu können müssen diese nach der Initialisierung zusätzlich beim *EventOutObserver* angemeldet werden (*EventOut* selber hat keine Methoden um den Wert einer Variable auszulesen), welcher dann in einem gesonderten Threat meldet wenn sich ein Wert verändert:

```
EventOutSFBool active = (EventOutSFBool)
  klicker.getEventOut("isActive");
isActive.advise(observer, this);
```

Nun kann das Applet mittels einer weiteren Klasse welche den *EventOutObserver* implementiert bei jeder Veränderung benachrichtigt werden:

```
public class MyObserver implements EventOutObserver{
public void callback(     EventOut value,
                double timeStamp,
                Object data)
          {
// cast value into an EventOutSFVec3f and use it
          }
        }
```

5.4 Zusammenfassung

Der VRML Standard stellt eine umfassende Anzahl von Datentypen, Klassen und Schnittstellen zur Verfügung um komplexe Aktionen mittels höherer Program−miersprachen ausführen zu können oder es dem Anwender zu ermöglichen mittels externer (Steuerungs−)Elemente das Geschehen innerhalb der virtuellen Welt abzufragen oder direkt darauf Einfluß nehmen zu können. Und das alles ohne die bestehenden Standards umgehen zu müssen. Die Erweiterung mittels zum Bei−spiel externer Java−Applikationen eröffnet wesentlich mehr Möglichkeiten als dies der VRML Standard allein vermag. Beispiele wären die Berechnung von dreidimensionalen Fraktalen, die Darstellung dynamischer Diagramme in 3D sowie die vernetzung von Welten mittels Java−Sockets. Letzeres Beispiel, als Hauptbestandteil dieser Diplomarbeit, soll, unter Berücksichtigung der bereits vorhandenen Lösungsansätze und −vorschläge, an einem eigenen Projekt im nächsten Kapitel dargestellt und erläutert werden.

6 Entwurf einer virtuellen Umgebung mittels Java und VRML

Dieser Abschnitt beschreibt das für diese Diplomarbeit entwickelte und implementierte Projekt einer verteilten virtuellen Welt mittels VRML und Java. Dazu wurde ein bereits vorhandenes Konzept (VNET) mit eigenen Applikationen erweitert und so den Anforderungen an die Aufgabenstellung aus Abschnitt 1 gerecht zu werden. Zunächst soll die Benutzeroberfläche und die dahinter stehenden Applikationen erläutert werden bevor im nächsten Abschnitt die technischen Hintergründe der Implementation aufgezeigt werden.

6.0.1 Technische Voraussetzungen

- Java fähiger Webbrowser sowie EAI−fähiges VRML Plugin
- Eingesetzte Browser MS−Internet Explorer mit VRML Erweiterung sowie Netscape 4.5 und Cosmo Player 2.1.
- Betriebsysteme: Windows 9x und NT sowie Macintosh und Irix (Cosmo Player − noch nicht getestet). Im Moment gibt es noch keine VRML Browser für Linux/Unix die die API einigermaßen vollständig implementiert haben.

Entwickelt wurde das Projekt auf Pentium II Rechnern mit 266 und 333 Mhz unter Windows 98 und NT 4.0. Als Entwicklungsumgebung wurde das Java Development Kit 1.1.7 für Windows für die Compillierung der Java Applets, sowie GNU Emacs 20.1 für Windows für das Schreiben des Java und VRML Quellcodes eingesetzt.

6.0.2 Kompatibilitätsprobleme

IE Explorer 4.0: Das Erkennen von Prototypen funktioniert nur sehr unzuverlässig was auf eine unvollständige Implementierung der EAI Spezifikation schließen läßt.

Netscape: Version 4.6 hat einen Bug in der EAI welcher verhindert das der Typ des verwendeten Browsers (Methode *getBrowser*) an Java Applets zurückgegeben wird. Dadurch ist das gesamte System nicht lauffähig. Es gibt keine Probleme mit Version 4.0−4.5. Außerdem stürzt Netscape manchmal beim Beenden es Systems ab, die Ursache liegt warscheinlich in einem Bug in der *Garbage Collection* im Cosmoplayer selbst und wurde in der VRML Newsgroup bereits ausführlich diskutiert, leider sind bisher aber noch keine Problemlösungen bzw. Bugfixes bekannt.

Cosmoplayer: Version 2.0 enthält noch einige Lücken in der EAI Implementation welche aber bei Version 2.1 beseitigt wurden.

6.1 Benutzeroberfläche

Das Browserfenster ist in drei Bereiche aufgeteilt, welche durch HTML−Frames gebildet werden. Im linken Frame befindet sich das Hauptfenster in welchem sich der VRML−Browser und das VNET−Chatapplet befindet. Der rechte Frame enthält den Bereich mit Statusanzeigen, sowie einen Bereich in dem Erläuterungen zu Objekten, Aufgabenstellungen sowie Links zu Hilfetexten, Tutorials und Hilfsprogrammen enthalten sind. Die einzelnen Seiten können über entsprechende Schalter umgeschaltet werden.

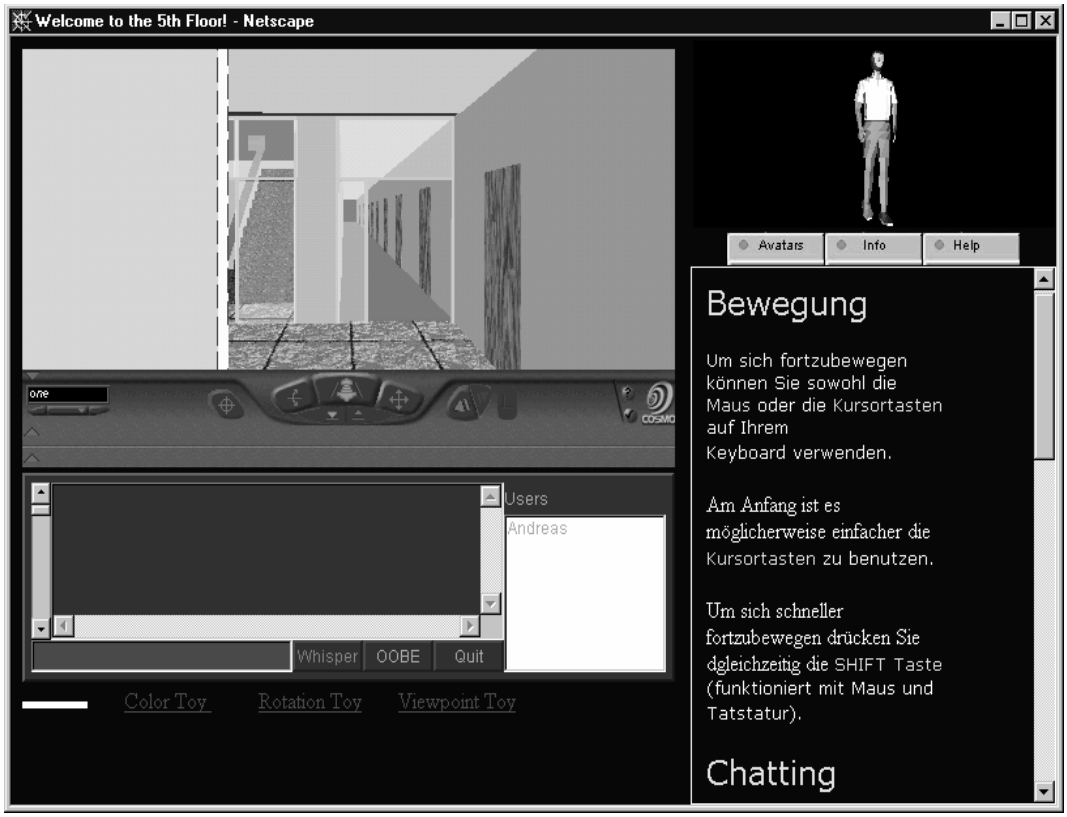

Abbildung 7: Das Browerfenster mit VRML− und Chatapplet

6.1.1 Räume

Im vorliegenden Projekt wurden, wie in der Aufgabenstellung vorgegeben, die Räume 5−01, 5−04/05 und 5−26 des Universitäts−Hauptgebäudes implementiert und gestaltet. Sämtliche anderen Räume der 5. Etage des Hauptgebäudes wurden ebenfalls eingebracht, stellen aber nur Platzhalter dar welche weder verlinkt noch eingerichtet sind. Dies kann selbstverständlich auch nachträglich erfolgen, z.B. in einem späteren Projekt oder Praktikum. Die dazu erforderlichen Maßangaben, um weitere Objekte maßstabsgerecht in die Räume einfügen zu können, befinden sich im Anhang C.

6.1.2 Vernetzte Objekte (VOP−Objekte)

VOP−Objekte (*VRML−Object−Package*−Objekte), welche im Abschnitt 7 beschrieben werden, werden mittels des entwickelten Java−Applets miteinander verlinkt und stellen somit veränderbare Objekte dar, welche sich für alle Benutzer jeweils im gleichen Zustand befinden. Verändert sich der Zustand eines Objektes (z.B. das Öffnen einer Tür) geschieht dies automatisch auch in allen anderen angeschlossenen Welten. Im vorliegenden Projekt geschieht das Auslösen von Ereignissen durch einen einfachen Mausklick auf das entsprechende Objekt. Andere Aktionen wie zum Beispiel das Verschieben oder Skalieren von Objekten direkt durch den Benutzer selbst sind noch nicht vorgesehen, können aber durch−aus als Folge von Mausklicks emuliert werden. Ein Beispiel wäre ein spezielles Kontrollpanel welches in den entsprechenden Raum gestellt wird und mittels eines *Switch* Knotens eine Folge von unterschiedlichen Ereignissen simuliert.

6.1.3 HTML−Seiten

Neben der eigentlichen virtuellen Welt wird dem Benutzer ein umfangreiches Hilfesystem auf HTML Basis zur Verfügung gestellt, welches in einem zusätzli−chen HTML−Frame neben dem VRML−Browser plaziert wird. Die Plazierung von Text in der virtuellen Welt selber sollte nur sparsam erfolgen, da einerseits die Qualität der verwendeten Schriften nicht besonders hoch ist, andererseits zusätzlich von verschiedenen Faktoren wie Transparenz, Farbe, Beleuchtung und Abstand abhängt. Außerdem ist das Lesen im dreidimensionalen Raum stark gewöhnungsbedürftig, da ja nicht automatisch davon ausgegangen werden kann das sich der Betrachter immer im optimalen Winkel vor der Schrift befindet. Neben allgemeinen Hinweisen zur Benutzung des Browser und der Navigation in der virtuellen Welt können dort auch Auswahlmenüs für die Auswahl eines ent−sprechenden Avatars bereitgestellt werden, ebenso wie Dokumentationen zu VRML Objekten oder Texte innerhalb einer Guided Tour. Im Praktikumsbetrieb ergibt sich weiterhin die Möglichkeit an dieser Stelle Aufgabenstellungen von einem schwarzen Brett abzurufen. Im momentanen Stadium des Projekts sind fol−gende Dokumente abrufbar:

- Begrüßungstexte und kurze Einführung
- Hilfesystem zur Navigation
- Linklisten
- umfangreiches VRML Hilfesystem (Beschreibung der Syntax und Nodes)
- einige Beispielaufgaben
- Impressum

6.1.4 Werkzeuge

Im Hilfepanel des rechten Frames sind verschieden VRML Werkzeuge abrufbar die u.a. die Zusammenhänge der verschieden Parameter in Transformation und Appearance Nodes deutlich machen. Auch wenn diese Tools selbst VRML−Code erzeugen so sind sie doch keine VRML−Entwicklungswerkzeuge im eigentlichen Sinne und daher nicht zur Erstellung von VRML−Szenarien geeignet. Der Praktikumsteilnehmer hat sich also nach wie vor mit den Elementen von VRML auseinanderzusetzen.

Tiny−3D
Dieses VRML−Entwicklungstool ermöglicht es mittels Schiebereglern die Einstellung der Parameter von Transformation, Rotation und Farben an einfachen Flächen, Kugeln oder Kegeln vorzunehmen. Das Tool erscheint nach dem anklikken des entsprechenden Feldes in einem separaten Fenster.

Color−Changer
Dieses unter GPL veröffentlichte Tool ermöglicht die komplexen Zusammenhänge von Oberflächengestaltung, Beleuchtung und Farben in VRML durch einfaches Ausprobieren überschaubarer zu machen. Neben der Einstellung von Farbe und Transparenz kann auch Anzahl, Typ und Richtung von Lichtquellen sowie deren Parameter wie Lichtstärke, Diffusion usw. eingestellt werden. Auch hier erscheint das Tool nach dem Anklicken in einem separaten Fenster.

6.2 Benutzung

6.2.1 Login/Logout

Aufgrund der Struktur von VRML als offener Webstandard und dessen Browser ((abschaltbare Kollisionsüberwachung) ist ein genereller passwortgeschützter Zugang zur virtuellen Welt im Moment nicht vorgesehen. Allerdings ist es möglich die Dateien selber auf einem passwortgeschützen Server unterzubringen so das hierdurch der Zugang eingeschränkt werden kann. Beim Betreten der Eingangsseite kann der Besucher zunächst auswählen, ob er nur lokal oder im Verbund mit anderen Browsern arbeiten möchte. Bei letzterem werden der Objektclient und VNet−Client gestartet. Das Einloggen in den Objektserver erfolgt automatisch, in den VNET−Server nach Eingabe eines Benutzernamens und der Auswahl eines Avatars. Ein Wechsel zwischen lokalem und verbundenem Modus während des laufenden Betriebs ist im Moment noch nicht möglich.

6.2.2 Bewegung in der virtuellen Welt

Standardmäßig ist in der Rootdatei (der obersten Datei der VRML−Struktur) der Bewegungsmodus *walk* eingestellt, welcher nur Bewegungen zu ebener Erde zuläßt. Dies kann auch durch den Benutzer während des Betriebs nicht verändert werden. Durch entsprechende Modifikation des *NavigationInfo* Nodes mittels eines Skripts kann aber durchaus auch im laufendem Betrieb zwischen den einzelnen Bewegungsmodi gewechselt werden. Vorstellbar wäre z.B. ein Raum in dem eine Menüleiste dem Benutzer die Möglichkeit bietet verschiedene Bewegungsmodi auszuprobieren.

6.2.3 Kommunikation

Die Kommunikation der Besucher untereinander erfolgt entweder über das eingebaute Chatapplet von VNET oder durch den Einsatz externer Programme welche eine Übertragung von Bild und Ton ermöglichen (siehe nächster Abschnitt).

6.2.4 Einsatz von externen Programmen

Standardmäßig sind in den VRML Browsern nur das Anzeigen von .JPG und PNG Bildern sowie die Wiedergabe von MPEG Layer I Filmen vorgesehen, wobei die Möglichkeiten der Steuerung nach wie vor stark eingeschränkt sind (kein Vor−/Rücklauf). Für den Nachfolger von VRML 2.0, Web3D bzw. VRML 200x, werden zwar erweiterte Möglichkeiten in der Umsetzung von Multimediaelementen angekündigt, voraussichtlich wird es aber noch mindestens 2 Jahre dauern bis dieser Standard aktuell sein wird. Bis dahin ist im vorliegenden Projekt der Einsatz von externen Applikationen vorgesehen, die mittels des HTML Browsers gestartet werden können. Da Java nicht ,wie ursprünglich vorgesehen, in der Lage ist Nicht−Java Programme selbständig zu starten wird hier eine andere Lösung vorgeschlagen.

Die Einbindung von externen Programmen erfolgt entweder

- als eingebettete Applikation in einer HTML Datei welche in einem benachbartem Frame (oder neuem Fenster) geladen wird, oder

- durch das Laden einer HTML Datei, welche einen Link zu dem entsprechenden Programm enthält, in einem unsichtbaren Frame. Der Browser wird dann, evtl. nach einer Sicherheitsabfrage, das entsprechende Programm starten sofern es auf dem lokalem Rechner vorhanden ist.

Die genaue Beschreibung der Syntax findet sich im Abschnitt 7.1.2 über die Parameter des Objekt Prototypen. Wichtig ist das alle Namen in Anführungszeichen geschrieben werden müssen, aber keine Trennzeichen verwendet werden, ebenso ist darauf zu achten das die Namen case−sensitive sind!

6.3 Möglichkeiten zur Geschwindigkeitssteigerung

Ein häufiger Nachteil von dreidimensionalen Welten ist, das ab einer bestimmten Größe der virtuellen Welt der durch den hohen Rechenaufwand verursachte Geschwindigkeitsabfall sich sowohl auf die Navigationsfähigkeit, als auch das virtuelle Empfinden negativ auswirkt. Letzteres läßt sich anhand der vom Browser gemessenen Framerate sogar grob bestimmen, wie eine im VRML Discussion Board veröffentlichte Liste von Rick Carey zeigt:

Frames pro Sekunde	**Effekt**
20−60	Perfekte Illusion, volle 3D−Erfahrung, traumähnliche Klarheit
12−20	Akzeptable Illusion von 3D, gut genug
8−12	Bedienbare 3D−Schnittstelle, unterbewußte Frustra−tion, schiffsgleiche Navigation (Verzögerungen), OK für Design−/Expertentool
<8	Nicht−verwendbare 3D−Schnittstelle, schlechte Navigation, bewußte Frustration, benötigt andere UI−Techniken (z.B. Myst)

Im Vordergrund der Betrachtungen steht dabei die Tatsache das Objekte, oder Teile von Objekten, welche sich vom Standpunkt des Betrachters nicht in dessen Blickfeld befinden, nach Möglichkeit nicht berechnet werden sollten und damit Rechenzeit eingespart wird. Einige der VRML−Browser der neueren Generation (Cosmoplayer 2.x, Cortona) haben bereits entsprechende Algorithmen implemen−tiert, diese kommen aber nur richtig zum Tragen wenn auch der Aufbau/die Kon−struktion der virtuellen Welt entsprechend durchgeführt wird. Die wichtigsten dieser Techniken und die daraus resultierenden Ergebnisse im vorliegenden Pro−jekt sollen im folgenden aufgezeigt werden:

Reduzierung der Polygone: In [2] wird empfohlen die Anzahl der Polygone einer Szene auf ein Mindestmaß zu beschränken, um sowohl eine Steigerung der Geschwindigkeit als auch eine Akzeptanz für Browser und PCs, welche nur eine begrenzte Anzahl von Polygonen darstellen können, zu ermöglichen. Als Richt−wert werden hier maximal 2000 Polygone genannt, dieser Wert dürfte sich aber durch die inzwischen wesentlich leistungsstärke Hardware inzwischen weiter nach oben verschoben haben. Detaillierte Ansichten sollten nach Möglichkeit immer mit entsprechenden Texturen kombiniert oder sogar vollständig ersetzt werden.

Instanziierung: Wird ein Objekt (oder Teil eines größeren Objekts) in eine Szene mehr als nur einmal verwendet ist es sinnvoll diesem Objekt mittels DEF einen Namen zuzuweisen um später Kopien dieses Objektes anfertigen zu können deren Quellcode lediglich aus einer Referenz zum definierten Objekt besteht:

```
DEF alu Material {
            ambientIntensity 0.3
            diffuseColor 0.3 0.3 0.3
            specularColor 0.7342 0.7 0.83421
            shininess 0.1
         }
#linke Wand
Transform {
   translation −1.0 −0.35 −0.2
   rotation 0 1 0 1.57
   children  DEF wall1 Shape {
      appearance Appearance {
         material USE alu
      }
      geometry Box {size 1.7 2.2 0.1}
   }
}

#rechte Wand
Transform {
   translation 1.0 −0.35 −0.2
   rotation 0 1 0 1.57
   children USE wall1
}
```

Dieses Beispiel zeigt neben der oben beschriebenen Mehrfachverwendung von Objekten noch eine weitere Möglichkeit zur Verwendung des DEF Konstrukts, nämlich die vorherige Definition von Parametern (in diesem Fall Material) und anschließende Verwendung innerhalb des Nodes. Diese Verfahren hat den Vorteil das komplexe Parameter mit langen Fließkommawerten nur einmal berechnet und abgespeichert werden müssen.

LODs: Der Level Of Detail Node (LOD) zeigt die in ihm eingebetteten Objekte (LOD ist ein Group−Node) abhängig vom Abstand des Betrachters an. Dadurch ergibt sich für den Programmierer die Möglichkeit, komplexe Objekte durch ein−facher strukturierte Objekte ersetzen zu können, wenn sich der Betrachter ent−sprechend weit davon entfernt hat. Der VRML Browser selber besitzt keinerlei Algorithmen welche die Detailtreue von Objekten abhängig von Abstand des Betrachters festlegen, so wie dies in der realen Welt durch die begrenzte Auflö−sung des menschlichen Auges geschieht. Dies bedeutet aber auch, das bei stark strukturierten Objekten jedes winzige Detail neu berechnet werden muß, auch wenn der Betrachter selbst dieses Detail aufgrund seiner Größe selbst nicht mehr erkennen kann. In [2] wird empfohlen, das ein LOD Node mindestens 3 Objekte unterschiedlicher Detailtreue enthalten soll, dagegen wurden im vorliegenden Projekt jeweils nur zwei Objekte verwendet. Der Grund liegt in den geringeren räumlichen Dimensionen, in welchen sich der Betrachter bewegt, und die durch die Abmessungen der Räume und Korridore fest vorgegeben sind, so das sich der Aufwand eines zweiten Platzhalter−Objekts nicht lohnt. Als Beispiel sei die

Implementierung des Raums 5−26, welcher zahlreiche strukturierte Elemente wie Monitore, Tische usw. enthält,aufgezeigt:

```
#Raum 5-26

Transform {
  translation  3.0 0.0 -3.5
  children [
     LOD {
       center 0 0 0
       range [ 7 ]
       level [
        Inline { url "room5-26.wrl" }
         Inline { url "room5-26d.wrl" } #leerer Raum
         ]
      }
      ]
   }
```

Der Raum selber ist als *Inline* Node in das Hauptdokument eingefügt. Da der Browser solche Nodes inklusive aller untergeordneten Objekte als einzelnes Objekt betrachtet und rendert, ergeben sich Probleme das Verfahren zur Geschwindigkeitssteigerung, wie das später beschriebene *spartial partitioning*, hier nicht angewendet werden können. Statt dessen sorgt der LOD Node dafür, das sich im Normalfall (wenn der Betrachter sich außerhalb des Raumes bewegt) nur eine leere Hülle an dieser Stelle befindet. Die dadurch erzielte Geschwindig−keitssteigerung ist beachtlich (etwas 10 Frames/s mehr wenn sich der Betrachter in der Nähe des Raumes aufhält), allerdings neigt vor allem der Cosmo Player zu mehr oder weniger starken Ruckeln wenn zwischen den einzelnen Zuständen umgeschaltet wird.

Verschachteln (nesting) Um festzustellen ob ein Objekt gerendert werden muß oder nicht, überprüft der Browser zunächst welche Objekte sich generell in sei−nem Sichtbarkeitsbereich befinden, und in welcher Hierarchie diese zueinander stehen. Danach werden alle Objekte, welche sich sowohl im Sichtbarkeitsbereich des Benutzers als auch gleich− oder unterhalb der entsprechenden Hierarchiee−bene befinden, gerendert − egal ob diese tatsächlich sichtbar sind oder z.B. durch eine Mauer verdeckt werden. Das Zusammenfassen von kleineren Objekten zu Gruppen und Untergruppen mittels der entsprechenden Group−Nodes, wie z.B. Transform, Group usw., erhöht die Körnigkeit der Szenerie. Das bedeutet das z.B. Objekte, die sich innerhalb eines anderen Objektes befinden, zu einer Unter−gruppe des sie umgebenen Objektes gemacht werden, und somit vom Browser gesondert betrachtet werden können.

Räumliche Partitionierung (spatial partitioning):
Prinzipiell ist der VRML−Browser nicht in der Lage größer Objekte in kleinere Bestandteile aufzulösen, wenn dies vorher nicht durch den Programmierer selbst getan wurde. Das heißt, ein Objekt welches aus einem Stück gefertigt wurde, wird immer komplett gerendert, egal ob es vom Standpunkt des Benutzers voll−ständig oder nur teilweise zu sehen ist. Ein oft verwendetes Beispiel (so auch in [2] und [4]) ist ein Gebäude welches wie ein „H“ geformt ist:

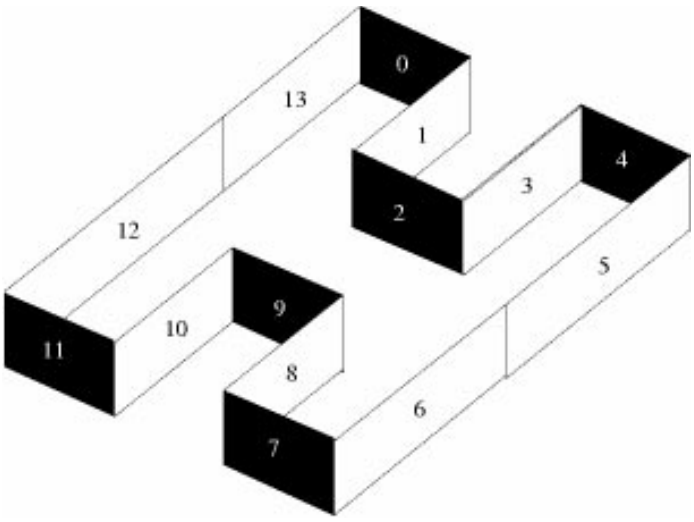

Abbildung 8: Teilung eines größeren Objekts in Unterobjekte

Implementiert man dieses Objekt als einen einzelnen Körper (z.B. mittels *indexedFaceSet*) so wird immer das gesamte Objekt gerendert werden müssen. Befindet sich der Betrachter beispielsweise in der Ecke welche von den Wänden 10, 11, und 12 umschlossen wird so sind dann die Teilbereiche 2 bis 9 von diesem Standort aus nicht sichtbar, müssen aber vom Browser trotzdem bei jeder Bewegung neu berechnet werden. Daher ist es wesentlich effektiver die einzelnen Wände auch als einzelne Objekte zu implementieren und diese dann auch in eine entsprechende Hierarchie einzubetten, wie das in [2] beschrieben wurde:

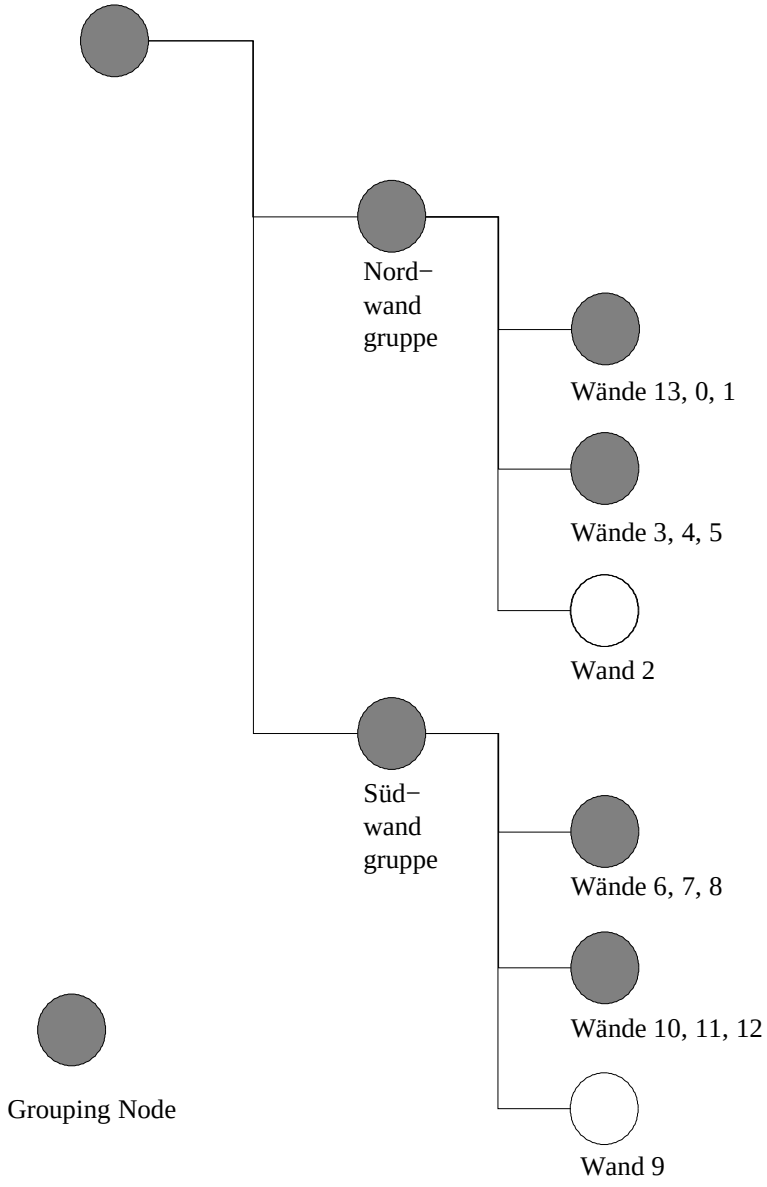

Abbildung 9: Anordnung der einzelnen Gruppenknoten

7 Implementation des Projekts in VRML und Java

7.1 Entwicklung eines eigenen VRML−Prototypen

7.1.1 Aufbau

Bedingung: Da eine spätere Modifikation des Prototyps auch eine Modifikation des Parsers erfordern würde (was unerwünscht ist), sollte dieser alle auch in der Zukunft zu erwartenden Einsatzmöglichkeiten berücksichtigen. Daher werden Applikationen die im aktuellen Standard noch nicht unterstützt werden (z.B. Streaming Audio und Video) als extern definiert und das dazu entsprechende Programm ausgeführt. Der Prototyp selber stellt nur das Grundgerüst für einen einfachen Touch−Sensor dar, die Struktur der Parameterübergabe orientiert sich am Anchor Node (URL als neu zu ladendes (HTML) Dokument, Parameterrumpf mit Angabe des Zielframes und weiteren Übergabeparameter). Übertragen wird lediglich der Name des Objekts, bei dem ein Ereignis eingetreten ist. Alle weiteren Aktionen (Animationen, Start von Applikationen) werden lokal im Objektrumpf beschrieben und ausgeführt. Diese Vorgehensweise hat den Vorteil, das nur eine Datenübertragung im Falle der Aktivierung/Deaktivierung eines Objekts durchgeführt werden muß, und das der Datenfluß durch das Fehlen von Parametern (die Parameter werden einmalig mit dem VRML−Hauptdokument geladen und stehen dann jederzeit zur Verfügung) minimal gehalten werden kann.

7.1.2 Parameter

Datentyp	**Name**	**geparst**	**Bemerkung**
SFVec3f	location	nein	Standort
SFBool	extern	ja	definiert URL als externes Programm
SFString	URL	ja	URL einer Website oder ext. Programms.
SFString	frame	ja	Zielframe für Dokument/Programm.
MFNode	children[]	nein	weitere Elemente/Skripte/Shapes
SFBool	isActive	ja	Objekt aktiv oder inaktiv (ein/aus)
SFTime	touchTime	ja	Zeitstempel bei Aktivierung
SFTime	touchTime2	ja	Zeitstempel bei Deaktivierung

location Parameter der Form {x y z} zur Zuweisung eines Standortes für den durch das Prototypen beschriebene Objekt.
Voreinstellung {0 0 0}. Nicht extern beeinflußbar.

extern Wenn TRUE dann wird der mit URL übergebene String nicht als Dokument sondern als extern auszuführendes Programm angesehen. Programmparameter können im frame−String übergeben werden.
Voreinstellung FALSE

URL Verweist auf ein lokales oder externes Dokument, welches beim Anklicken des Objekts geladen werden soll. Ist der Frame−String leer wird die aktuelle Welt durch das neue Dokument ersetzt (siehe auch Abschnitt 7.2.2.4).
Voreinstellung: Keine

frame Wenn nicht leer dann entweder Name des Zielframes oder Parameter für externes Programm (wenn *extern* gesetzt). Ist der Name des Zielframes nicht vorhanden wird ein neues Browserfenster geöffnet. Siehe auch 7.2.2.4.
Voreinstellung: Keine

children[] Hier können weitere Nodes wie Shapes, Gruppen usw. eingebunden werden. Kein externer Zugriff möglich.

isActive Wird TRUE (aktiviert) wenn das (nicht aktivierte) Objekt angeklickt, oder vom *Action−Handler* aktiviert wird. FALSE wenn das (aktivierte) Objekt wiederum aktiviert wird, oder ein Signal vom *Action−Handler* empfängt.
Voreinstellung: FALSE

touchTime Zeitstempel der bei der Aktivierung des Objekts gesetzt wird. Voreinstellung: keine

touchTime2 Zeitstempel der bei der Deaktivierung des Objekts gesetzt wird.
Voreinstellung: keine

7.1.3 Implementation

```
PROTO Object [

 exposedField   SFVec3f location 0 0 0
 field   SFBool extern
 field   SFString URL
 field   SFString frame
 field          MFNode  children  [ ]

 exposedField   SFBool  status FALSE
 eventIn        SFBool  set_status
 eventOut       SFBool  isActive
 eventOut       SFTime  touchTime
 eventOut       SFTime  touchTime2
 ]
{
   Transform {
      translation IS location
      children [
        Transform {
                children IS children
        }
        TouchSensor { isActive IS isActive
                      touchTime IS touchTime }

DEF script Script {
   eventIn  SFBool event IS set_status
   eventOut SFTime startTime IS touchTime
   eventOut SFTime startTime2 IS touchTime2
   field    SFBool state TRUE

   url  javascript:
    function event (value, timestamp) {
    if (value) {
        if (state) {
                startTime = timestamp;
                state = FALSE;
                }
        else {
                startTime2 = timestamp;
                state = TRUE;
                }
        }
   }
}
]
}
}
```

7.2 VRML−Object−Package (VOP)

7.2.1 Überblick

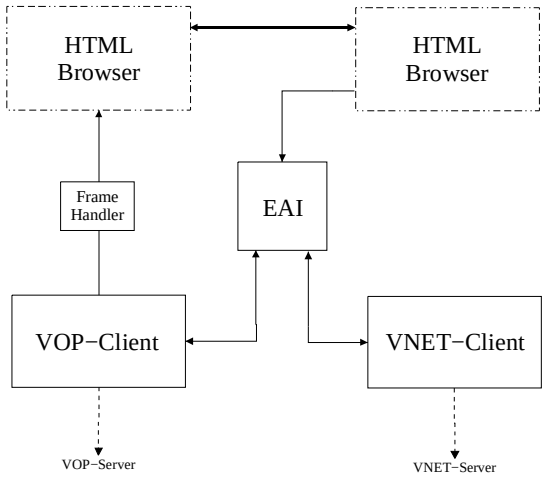

Abbildung 10: Der Aufbau des Clients

Eines der Hauptziele der Diplomarbeit ist die Entwicklung und Bereitstellung eines Programmpaketes welches es ermöglicht, Objekte in der virtuellen Welt mittels Java−Socket miteinander vernetzen zu können. Insbesondere sollen Schnittstellen bereitgestellt werden die es dem Anwender ermöglichen, Modifikationen an der virtuellen Welt problemlos und ohne Kenntnis des inneren Aufbaus der Java−Applikationen durchführen zu können. Weiterhin bildet das Paket eine Schnittstelle zu externen Anwendungen, die mittels VRML−Objekten gesteuert werden können.

Im einzelnen ist vorgesehen:

- Bereitstellung eines Schalters der Objekte mittels Java Sockets netzwerküber−greifend verbindet und steuert
- Semi−automatisches Erkennen von Objekten die neu eingebunden wurden
- Laden von HTML−Dokumenten in einem anderem HTML−Frame oder neuem Browserfenster
- Bereitstellung einer Schnittstelle zu Steuerung der virtuellen Szene mittels Java Buttons
- Aufruf von externen Programmen durch den HTML Browser wenn eine Anwendung als *extern* deklariert wurde.
- Implementation dieser Eigenschaften in einem eigenen VRML−Prototypen.

7.2.2 Client

7.2.2.1 Übersicht

Aufgrund der Sicherheitsbestimmung in Java ist eine Client/Server Lösung nur über ein eingebettetes Java Applet möglich.

Der Client besteht aus:

1. Proto Parser Applet, welches auch die Klassen

2. EventHandler und

3. Frame Handler enthält.

4. Sende/Empfänger Applet

Durch die Trennung EventHandler und Sende/Empfangsapplet kann der Client auch ohne Objektserver im Einzelbetrieb arbeiten. Ein direkter Client−Client Betrieb ist aber nicht möglich.

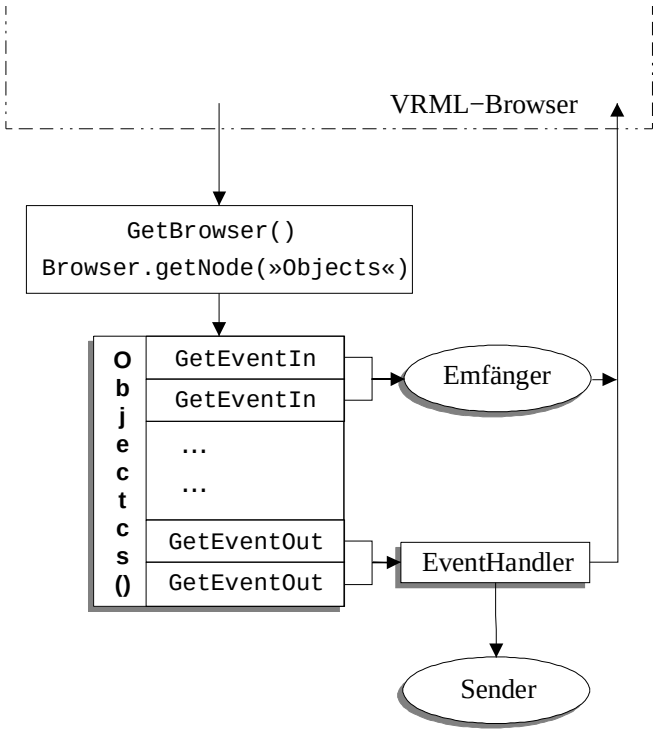

Abbildung 11: Aufbau des VOP−Applets

7.2.2.2 Proto−Parser−Applet

Die Klasse *protoParser* hat die Aufgabe alle im Abschnitt 7.1.2 angegebenen Parameter aus dem Haupt−VRML−Dokument für jedes Objekt auszulesen, zu speichern, und dem EAI als I/O Schnittstellen zur Verfügung zu stellen. Dabei wird auch die richtige Syntax der Parameter überprüft und eventuelle Fehler dem Benutzer mitgeteilt. Der Protoparser steht als Applet direkt mit dem VRML EAI in Verbindung und bildet die Verbindung zwischen VRML−Welt, Event− und Frame Handler sowie den Java Sockets. Der Parser ist die erste Klasse die nach der Initialisierung aufgerufen wird, und initialisiert ihrerseits alle weiteren Klas− sen nach erfolgreicher Ausführung des Parsings.

Im einzelnen sind folgende Aufgaben durch das Applet zu erledigen:

- Erkennung und Einbindung des Browsers
- Einlesen der Objektliste
- Bereitstellung von Speicherplatz und Schnittstellen für jedes Objekt
- Starten des Frame Handler Applets
- Starten des Event Handlers
- Starten des Sende/Empfangs Applets und Abrufen der Login−Informationen

Ein Objekt ist immer ein vordefinierter Prototype der Form *VRMLObject*, welcher am Beginn des Haupdokuments (Rootdatei) definiert ist. Die Objektliste ist ein *MFString[]* im VRML−Hauptdokument, in welcher alle zu vernetzenden Objekte der virtuellen Welt aufgeführt sind. Ein direktes Kennzeichnen dieser Objekte ist nicht möglich, da das EAI laut Spezifikation eine solche Methode nicht imple− mentiert hat, und der zur Kennzeichnung verwendete DEF Konstruktor zur Namensgebung auch bei anderen Anwendungen, z.B. Skripten verwenden wird, und somit keine Möglichkeit der Unterscheidung gegeben ist. Im vorliegenden Fall genügt es aber den Namen eines neu eingefügtes Objekts in die Objektliste einzutragen um es für den Parser sichtbar zu machen.

7.2.2.3 VRMLObjectType

Dieser Datentyp ist das Java−Abbild des ObjectPrototype in VRML. Implementiert und initialisiert durch den Proto−Parser wird Speicherplatz für folgende Daten reserviert:

```
String           name;
String[]         url = new String[1][20];
Node             node;
boolean          status = false;
EventInSFBool    setStatus
String[]         frame = new String[1];

EventOutSFBool isActive
EventOutSFBool status_changed
EventOutSFString urlstring
EventOutSFString framestring
```

Weiterhin werden folgende Methoden bereitgestellt:

```
String getName() { return name; }
       //Gibt den Namen des VRML−Objekts zurück

public String getURL()
      // Gibt die URL zurück

public void set_status(boolean status)
      // Schaltet zwischen den Zuständen um

public void restart(Browser browser,
                    EventOutObserver observer)
      //Initialisiert die Datenfelder
```

7.2.2.4 Frame−Handler−Class

Analog dem Anchor Node soll der Objekt Prototype in der Lage sein, andere Dokumente (Dokumente können andere HTML Seiten, Bilder, Töne oder auch VRML−Welten sein) in das aktuelle Browserframe oder in ein anderes Frame zu laden. Zu diesem Zweck werden der Klasse zwei Parameter übergeben:

```
MFString adresse[]
MFString frame[]
```

Ist der *frame*−String leer, wird das im Adressfeld angegebene Dokument im selben Frame geladen, in der sich auch die virtuelle Welt befindet. Das heißt aber auch, das diese durch das neue Dokument ersetzt wird. Dies könnte z.B. sinnvoll sein, falls zwischen verschiedenen Welten gewechselt werden soll. Enthält der Frames−String den Namen eines Frames der nicht vorhanden ist, so wird ein neues Browserfenster geöffnet, in welchem dann das Dokument geladen wird

20 Die Java−Methode *loadURL* des VRML EAI verlangt einen String der Form *MFString* (also Feld), so das hier ein Feld mit nur einem Eintrag reserviert wird. Das gleiche gilt für die zu übergebenden Parameter an *loadURL*, welche hier als Variable *frame* als einzelliger *MFString* auftaucht.

(obwohl dies eine übliche Methode ist um ein neues Browserfenster zu öffnen sollte man dafür besser JavaScript einsetzen, da dann die Höhe und Breite des neuen Fensters voreingestellt werden kann). Im Gegensatz zum Anchor Node wird der Parameterstring aber ausschließlich zur Benennung des Zielframes ver−wendet.

7.2.2.5 Sender/Empfänger−Applet (VRMLObjectClient.class)

Durch die Restriktionen von Java−Klassen können Sockets innerhalb von HTML Dokumenten nur als Applets ausgeführt werden. Das Client−Applet besteht im wesentlichen aus zwei Teilen:

- der Sender der direkt mit der *action−handler−class* verbunden ist und alle eintretenden Ereignisse sofort an den Objektserver übermittelt
- der Empfänger der als eigener Thread ausgeführt wird und ankommenden Nachrichten an das für sie bestimmte Objekt weiterleitet.

Für die Datenübertragung werden einfache Java−Sockets aus der Java−Package *java.net.Socket* eingesetzt:

```
socket = new Socket(url.getHost(), 8765);
System.out.println
Connecting to Objectserver.....");
in = new DataInputStream(socket.getInputStream());
out = new PrintStream(socket.getOutputStream());
} catch (IOException e)
```

7.2.2.6 Protokoll

Um den Datenstrom zwischen Server und Client gering zu halten werden Ereig−nisse nur dann übertragen, wenn sie eintreten. Es gibt keine Status oder Kontroll−mechanismen, ausgenommen den internen Datenaustausch zwischen den Sockets von Client und Server, und deswegen auch keinen stetigen Datenfluß, was die benötigte Bandbreite auf ein Minimum reduziert. Jedes Ereignis identifiziert sich über den Namen des Objekts das es ausgelöst hat, und welcher ja aufgrund der VRML Spezifikation innerhalb eines VRML Dokuments eindeutig sein muß. Je nach Anordnung der Objekte innerhalb der Hierarchie kann somit ein Objekt mehrere Ereignisse gleichzeitig auslösen.

7.2.2.7 Methoden

```
boolean connect(URL url, VRMLobjectType[]
  objects, Frame frame)
```

Diese Methode gibt als Rückgabewert den Status der Verbindung zum Server an (TRUE=connected, FALSE=lokal). Kommt keine Verbindung zustande wird durch den ExceptionHandler ein Dialogfenster geöffnet das es dem Benutzer ermöglicht zu versuchen, die Verbindung erneut herzustellen oder lokal zu arbeiten.

```
void SendNode(String nodeName)
```

Diese Methode sendet den als String übergebenden Nodename an den Objektserver (siehe 7.2.2.6).

7.3 Objekt−Server

7.3.0.1 Übersicht

Die Aufgabe des Servers ist es alle von den Klienten eintreffenden Ereignismeldungen in einer Datenbank zu speichern bzw. zu löschen, und diese gleichzeitig an alle anderen Clients weiter zu verteilen. Die interne Datenbank gewährleistet, das sich alle angeschlossenen Welten immer im gleichen Zustand befinden, egal zu welchem Zeitpunkt sich der Benutzer ein− und auslogt. Gleichzeitig werden die Benutzeraktivitäten in einer Datei aufgezeichnet.

7.3.0.2 Aufbau/Bedienung

Der Objektserver besteht neben dem Hauptprogramm aus den Klassen *Objects−Server.class*, *connection.class* und *ObjectDB.class*. Der Server wird mit dem Befehl

```
java ObjectServer [ −o filename]
```

gestartet und sendet/empfängt danach auf Port 8765. Durch den optionalen Parameter *−o filename* kann eine Datei spezifiziert werden, in die alle Ausgaben des Programms geschrieben werden sollen. Fehlt dieser Parameter werden alle Meldungen auf die Standardausgabe ausgegeben.

7.3.0.3 Sockets (connection.class)

Sendet ein Client eine Verbindungsanfrage an den Server, wird durch die Klasse *connection* ein eigener Thread für diese Verbindung erzeugt, und die Verbindung hergestellt. Empfängt der Server auf Port 8765 ein Ereignis von einem der angeschlossenen Clients wird dieses an alle Clients, einschließlich dessen von dem die Nachricht gekommen ist (Echo), ausgesendet. Der Inhalt der Nachricht, einschließlich dem Namen des Absenders, wird auf der Standardausgabe ausgegeben, bzw. in die Protokolldatei geschrieben.

7.3.0.4 Datenbank/Ereignispeicher (ObjectDB.class)

Um zu gewährleisten, das sich alle angeschlossenen Welten unabhängig vom Zeitpunkt ihres Einloggens immer im gleichen Zustand befinden, enthält der Server einen einfachen Speicher welcher die eintreffenden Ereignisse aufzeichnet, um diese später abrufen zu können. Dabei kommt das Ein/Aus Prinzip zu Anwendung: Wird ein Ereignis empfangen welches bereits gespeichert ist wird dieses gelöscht, andernfalls gespeichert. Ein (später) einloggender Client empfängt zunächst alle noch bestehenden Ereignisse (Synchronisation) bevor er selbst Ereignisse senden kann.

7.3.0.5 Logging

Wie schon im vorherigen Abschnitt angedeutet können alle Aktivitäten welche über den Server laufen in einer Datei geloggt werden, dabei werden neben der Datum und Zeit die IP des Senders und die übertragenen Daten aufgezeichnet.

7.4 VNET und VOP

7.4.1 Grundaufbau der virtuellen Welt

```
#VRML V2.0 utf8
#
# First comes all the stuff used by VNet
#
PROTO VNetInfo [
    exposedField        MFString        avatarNames             []
      .....
 ] { Group {} }

    DEF ROOTNODE Transform {
    DEF BIGBOX ProximitySensor {
    DEF VIEWPOINT Viewpoint {
    DEF LIGHT DirectionalLight {        # First child
    DEF VNET VNetInfo {
      avatarNames [  Red Men   Red Woman   Blue Man   Blue
         Woman   Custom  ]
      avatarURLs  [  RedMen.wrl   RedWoman.wrl
            BlueMan.wrl   BlueWoman.wrl    ]
      port                      8888
      root                      USE ROOTNODE
    }
  ] # end of children for world
  ROUTE VNET.isConnected TO LIGHT.on
}
#
#End of VNet Nodes. Now we continue with our own Object Proto
#

PROTO Objects [
    exposedField        MFString objects [  Object1"  Object2"
Object3" ]
        ]
DEF ObjectList Objects {}
PROTO Object
```

7.4.2 Erläuterung

Wie schon im Abschnitt 5.3 erläutert müssen sich laut EAI Spezifikation sämtliche VRML Elemente, auf welche mittels der EAI zugegriffen werden soll, im Haupt−VRML−Dokument (Rootdatei) befinden. Zunächst werden alle Elemente für den VNET−Client definiert und bereitgestellt sowie verlinkt. Anschließend wird der Object−Prototype (hier verkürzt dargestellt) definiert, danach kommt die Liste der vernetzten Objekte in Form eines Prototypen welcher nur einen einfachen *MFString* enthält. Danach wird der Prototyp und dessen Instanzen wie im Abschnitt 7.1.3 beschrieben definiert, anschließend folgt der normale VRML−Code, eingeschlossen aller Inline und Extern−Proto Referenzen. Es sei nochmals darauf hingewiesen das die Namen der Objekte case−sensitiv sind und nur einmal vergeben werden dürfen. Findet die EAI zwei gleichnamige Objekte vor wird gemäß Standard das letzte Objekt eingebunden.

8 VRML im Praktikumsbetrieb

Im Rahmen der Diplomarbeit soll der Einsatz dieser virtuellen Welt als Praktikumsversuch an der Universität untersucht und eingerichtet werden. Im folgenden sollen mögliche Aufgabenstellung bei einem solchen Praktikum aufgezeigt werden, die entsprechenden Lösungen sind ebenfalls im Anhang bzw. im Web zu finden:

8.1 Aufgabenstellungen

8.1.1 Umgang mit dem Browser:

Anhand einer Guided Tour lernen die Teilnehmer den richtigen Umgang mit den spezifischen Bedienungselementen kennen und erhalten so ein Gefühl für verschiedene Bewegungsmodi:

Aufgabe I

1. Begeben Sie sich auf die Homepage des Projekts und klicken Sie auf dem Menüpunkt: Browsersteuerung erlernen. Machen Sie sich anhand der Beispiele mit den Steuerungselementen des VRML−Browsers bekannt. Wichtig: Benutzen sie für alle Aufgaben den Netscape Browser ab Version 4.x (Windows und Mac, außer 4.6.0) mit Cosmoplayer 2.x VRML Plugin. Für die lokal zu erledigenden Arbeiten (ohne Java Unterstützung) kann auch der Internet Explorer ab Version 4.0 mit VRML Plugin verwendet werden.

2. Schalten Sie probeweise die Kollisionserkennung im Optionsmenü aus.

8.1.2 Schreiben von VRML−Objekten:

An praktischen Beispielen werden, ähnlich wie bei der Guided Tour, die Konstrukte von VRML Elementen und −aktionen erklärt und vorgeführt, aufgrund dessen dann der Teilnehmer eigene Objekte bauen und einbinden kann. Zur Durchführung stehen zwei Möglichkeiten zu Verfügung:

a. Der Teilnehmer erhält (wie im folgenden angenommen) ein eigenes Verzeichnis auf dem Praktikumsserver welches eine identische Kopie der virtuellen Welt enthält, auf die er auch schreibend zugreifen kann. Die Aufgabenstellungen beziehen sich dann auf den Raum 5−01 der entsprechend der Vorgaben auszugestalten ist. Vorteil dieser Lösung ist der einfach zu realisierende Passwortschutz, Nachteil ist das alle neuen Elemente nur lokal sichtbar sind.

b. Der Teilnehmer erhält einen eigenen virtuellen Raum der entsprechend der Vorgaben auszugestalten ist. Dieser Raum befindet sich als Inline Datei in einem Verzeichnis auf das nur der Benutzer schreibenden Zugriff hat. Vorteil dieser Lösung ist das die Objekte sofort für alle Benutzer der virtuellen Welt sichtbar werden.

Im folgenden werden nun Aufgaben zur Durchführung des Praktikums, geordnet nach Themengebieten, vorgeschlagen. Bis auf Aufgabe Nummer Eins (Einführung) sollten die Teilnehmer alle Aufgaben direkt in der virtuellen Welt am schwarzen Brett abrufen können.

Aufgabe II

Einführung: Für die Praktikumsarbeit wurde in ihrem lokalen Homeverzeichnis eine voll funktionsfähige Kopie der virtuellen Welt bereitgestellt welche sich im Verzeichnis vrml/ befindet. Dies ermöglicht Ihnen eigene Elemente in die virtuelle Welt einzubringen. Rufen Sie mit ihrem Browser die Datei index.html auf und wählen Sie ob sie nur lokal oder im Chatmodus arbeiten möchten. Ein Wechsel während der Arbeit ist im moment noch nicht möglich, dazu muß die Seite neu aufgerufen werden. Machen Sie sich mit der virtuellen Umgebung und dem Chatapplet vertraut. Zu Ihrer Hilfe sind im rechten Browserframe mehrere Menüpunkte mit Links zu Hilfeseiten und Tools vorhanden. Außerdem sind für die meisten Objekte Erklärungen vorhanden die durch anklicken auf das Hilfesymbol angezeigt werden.

1. Machen Sie sich anhand der Hilfetexte mit dem Aufbau von VRML vertraut. Entwickeln Sie anschließend einen einfachen virtuellen Stuhl unterVerwendung von Box Knoten und schreiben Sie den Quelltext in eine Datei mit Namen *stuhl.wrl* . Bitte verwenden Sie keine Indexed face Sets.

2. Binden Sie die Datei stuhl.wrl als Inline File in die virtuelle Welt (Raum 5−01) ein. Überlegen Sie sich welche Datei sie dazu modifizieren sollten um möglichst unkompliziert die Werte für den übergeordneten Transformknoten (welcher die Lage des Stuhls im Raum bestimmt) bestimmen zu können.

3. Reproduzieren Sie den Stuhl mehrfach (3−5 mal) und ordnen sie die »Kopien« entlang der vorhandenen Computertische an. Versuchen Sie bei der Reproduktion auch DEF−USE Ausdrücke anzuwenden.

4. Entwickeln Sie einen virtuellen Drehstuhl mittels IndexedFaces Sets welcher mindestens die doppelte Größe als normal haben soll. Binden sie diesen Stuhl wie gehabt in den Raum 5−01 ein (skalieren nicht vergessen!) und reproduzieren Sie ihn drei− bis viermal.

8.1.3 Texturen

Aufgabe III

- Ergänzen Sie den Raum 5−01 mit einem Poster an der rechten Wand, dazu steht ihnen die Datei *sgi.png* zur Verfügung.

8.1.4 Sensoren

Aufgabe IV

1. Modifizieren Sie mittels Grid−Sensoren einen oder mehrere Ihrer Drehstühle dahingehend das diese beim anklicken und festhalten der linken Maustaste durch den Raum geschoben werden können[21].

2. Erweitern Sie das Poster um einen Anchor−Node welcher beim anklicken die Homepage von SGI (www.sgi.com) in einem neuen Browserfenster läd. Hinweis zur Durchführung: Schauen Sie sich in der VRML Spezifikation den Aufbau des Anchor−Nodes an und geben Sie für Target den Namen eines nicht vorhandenen HTML−Frames an (z.B. neu). In solch einem Fall öffnet dann der HTML Browser automatisch ein neues Fenster.

8.1.5 Animationen

Aufgabe V

1. Entwerfen und implementieren sie ein einfaches regelmäßig schwingendes Pendel welches sie als Schreibtischschmuck auf den Tisch plazieren. Verwenden zur Realisierung der Animation JavaScript innerhalb des VRML−Quellcodes.

2. Zusatzaufgabe: Das (in diesem Fall) ruhende Pendel soll durch anklicken zum Schwingen gebracht werden bis es nach einiger Zeit von selber auspendelt.

8.1.6 Beleuchtung

Aufgabe VI

Bringen sie an der Decke des Raumes eine „Glühlampe“ (einfache Kugel) an und erzeugen sie mittels des *SpotLight* Nodes die Illusion einer Bestrahlung des Raumes durch diese Lampe. Das Licht soll dabei scheinbar vom Mittelpunkt der Lampe ausgehend zu den Wänden hin kreisförmig immer schwächer werden. Hinweis: Schauen Sie sich die Parameter des *SpotLight* Nodes an und finden Sie heraus mit welchem Parametern Winkel und Radius der Lichtstrahlen beeinflußt werden können.

21 Diese Aufgabenstellung könnte später unter Verwendung des Objekt−Prototypen weitergeführt werden, bei dem das verschieben von Objekten auch für die anderen Praktikumsteilnehmer sichtbar ist.

8.1.7 Prototyping

Aufgabe VII

In der vorliegenden virtuellen Welt werden die meisten Objekte mehrfach verwendet (z.B. Räume, Türen, Monitore), dies geschieht durch Benutzung des Inline−Nodes mit welchem die Quelldateien einzelner Objekte an beliebigen Stellen neu eingebunden werden können. Vorteil dieser Methode ist die einfache Implementierung und gute Übersichtlichkeit. Nachteil ist das die verwendeten Objekte in ihrem Aussehen nicht, oder nur im geringen Maße durch Skalierung, verändert werden können. Abhilfe schafft hier der Einsatz von selbstdefinierten Prototypen, an welche Parameter übergeben werden, die Auswirkungen auf Gesamt bzw. Teilbereiche haben können:

1. Entwickeln sie einen neuen Prototype Raum welcher in seinem Parameterfeld die Übergabe der Raumnummer und bis zu zwei Namen ermöglicht. Diese Werte sollen dann links neben der Tür in etwa 1,20m Höhe erscheinen. Zur Gestaltung des Raumes beachten Sie bitte die vorgegebenen Standards bezüglich Ausmaße und Ausrichtung[22].

2. Ersetzen Sie im Modul *5thfloor2.wrl* die Räume auf der rechten Seite durch Ihren Prototype. Versuchen Sie dabei nach Möglichkeit die Ausrichtung ihres Prototypen so zu gestalten das Sie nur die Inline−Ausdrücke ersetzen müssen ohne die Transform−Werte zu verändern.

3. Erweitern Sie Ihren Prototypen Raum um die Parameter *deep, width, alignDoor*. Die Parameter sollten folgendes bewirken:

Parameter	**Beschreibung**	**Default**
deep	Tiefe des Raums	7
width	Breite des Raums	3.56
alignDoor	Lage der Tür an der Frontseite (left, right, center)	center

4. Entwickeln Sie einen neuen Prototypen *door* welcher Parameter zur Darstellung der Oberfläche (Holz, Glas) enthält. Überlegen Sie sich wie der Prototype aufgebaut sein muß wenn als Parameter nur einfache Strings (*wood, glas*) aber keine Inline−Files zugelassen sind. Legen Sie als Defaultwert *wood* fest.

5. Erstellen Sie eine neue Datei mit dem Namen prototypes.wrl welches die beiden Prototypen *door* und *room* enthält. Verändern Sie die Datei *5thfoor2.wrl* so das der Zugriff auf den Prototypen *door* nunmehr ausschließlich durch

22 (Diese Standards werden in den HTML Hilfsdateien bzw. Objekterläuterungen beschrieben, und sind auch im Anhang C zu dieser Diplomarbeit einsehbar.

Benutzung von *prototypes.wrl* erfolgt. Das Inline Field *door.wrl* ist dabei durch einen Zugriff auf den Prototypen *door* zu ersetzen.

6. Erweitern Sie den Prototypen *room* um den Parameter *door* welcher das Aus−sehen der Tür bestimmt. Gestalten Sie den Prototypen so das dieser Parameter direkt an den eingebetteten Prototypen *door* weitergegeben wird.

9 Auswertung des Testbetriebes

Die Stabilität der verwendeten Programme und Applets kann als ausreichend bezeichnet werden. Während des praktischen Betriebs wurden keine Programm− bzw. Systemabstürze beobachtet, es traten lediglich vereinzelte Abstürze beim Neuladen (Reload) der Welt auf welche, wie schon im Abschnitt über Kompatibi−litätsprobleme angegeben, sich offenbar aus einem Bug in der EAI des verwende−ten VRML Browsers (Cosmoplayer) herleiten. Anfangs beobachtete Abstürze des VOP−Applets und ein damit verbundenes Einfrieren des Browsers konnte nach Modifikation des Quellcodes aufgrund von Hinweisen aus *comp.languages.vrml* vollständig beseitigt werden, Ursache war höchstwahrscheinlich wieder ein Bug in der EAI Implementation des Cosmoplayers, welcher die einfache Browserer−kennung durch die EAI gemäß dem Standard (*get.Browser*, siehe Abschnitt 5.3) fehlerhaft interpretiert. Die Server, sowohl VNET als auch der Objektserver, sind bisher auch über längere Zeiträume stabil gelaufen. Wie weit sich beide System skalieren lassen ist nicht bekannt. Test mit bis zu 10 Benutzern sind bisher aber problemlos verlaufen.

Ein wichtiger Punkt der abschließenden Beurteilung war auch die Geschwindig−keit der Browser bei Bewegung in der virtuellen Welt. Gerade hier gab es am Anfang einige Schwierigkeiten als die Implementation der einzelnen Räume einen gewissen Umfang erreicht hatte. Es zeigte sich, das die rein lineare Anord−nung der Elemente zu größeren Geschwindigkeitseinbußen führt, und somit die erreichbare Framerate ab einer bestimmten Anzahl von Objekten drastisch absinkt. Hier half schließlich die konsequente Anwendung der in Abschnitt 6.3 diskutierten Methoden zur Geschwindigkeitssteigerung, speziell das Partitionieren der gesamten Welt und die Nutzung von LODs zur Reduzierung der Anzahl der verwendeten Polygone.

Demgegenüber als unbefriedigend zu bezeichnen sind die Gesamtanforderungen welche an das System beim Nutzer zu stellen sind. Hier war ursprünglich vorge−sehen, eine Nutzung auch auf leistungsschwächeren Rechnern zu ermöglichen. Durch die Einbindung von Java sowohl auf Seiten der Applets (VNET und VOP) als auch innerhalb des VRML Browsers[23] sinkt die Leistungskurve selbst lei−stungsstarker Systeme wie die verwendeten Pentium II Rechner nach dem Starten rapide ab. Dies ist definitiv auf den Einsatz mehrerer virtueller Java Maschinen innerhalb des Browsers zurückzuführen. Eine Entlastung wie zum Beispiel durch die Verwendung von 3D Graphikkarten (wie am Diplomandenrechner erfolgt) ändert an diesem Zustand nichts. Es bleibt festzustellen das für einen einigerma−ßen flüssigen Betrieb die Systemanforderungen mindestens bei einem Pentium II System liegen (AMD nicht getestet) sollten. Spezielle 3D Grafikkarten sind mei−ner Auffassung nach nicht unbedingt erforderlich da das interne Software Rende−ring offensichtlich sehr effizient arbeitet, bei leistungsschwächeren Systemen (Pentium 166 o.Ä.) aber möglicherweise doch sinnvoll ist.

23 Der Großteil des Cosmo−Player Plugins ist in Java geschrieben worden

10 Zusammenfassung und Ausblick

Die vorliegende Diplomarbeit hat gezeigt das eine Vernetzung virtueller Welten in VRML durch Einbindung von Hochsprachen wie Java und JavaScript sowie die Verwendung vorhandener Lösungsansätze möglich ist, und durch Applikationen beliebig erweitert werden kann. Es wurde darauf Wert gelegt das weder ein VRML−Browser neu zu entwickeln war, noch das der Anwender Kenntnisse des Java−Quellcodes besitzen oder diesen neu compilieren muß, um neue externe Applikationen einzubinden. Es genügt in diesem Fall lediglich die Struktur des VRML−Prototypen zu kennen, um diesen dann entsprechend im VRML Quelltext einbinden zu können. Auch wenn der Server ausgefallen ist oder nur lokal gear−beitet werden soll funktioniert das System, dann selbstverständlich ohne Avatare.

Der VRML 2.0 Standard, so wie wir ihn heute kennen, wird voraussichtlich noch ein oder zwei Jahre bestehen, bevor die Spezifikationen zum neuen Web3D Stan−dard endgültig abgeschlossen sind und erste Browser verfügbar sein werden. Die vorliegende Arbeit trägt dem Rechnung und erlaubt bis dahin die Einbindung verschiedenster Anwendungen als externe Applikationen, welche im zukünftigen Standard ihren festen Platz haben. So wird die Integration der Vernetzung mittels Sockets ebenso in den zukünftigen Browsern implementiert sein als auch eine universelle Verwaltung von Avataren. Streaming Audio und Video werden ebenso verfügbar sein wie die textbasierter Chat oder Audiokonferenzsysteme, universelle Datenbankanbindungen und das alles ohne externe Anwendungen ein−binden zu müssen. Aber auch dann wird das vorliegende System, dank Abwärts−kompatibilität, noch vollständig lauffähig sein.

Literatur und Linkverzeichnis

[1] Hase, H.−L.: Dynamische virtuelle Welten mit VRML 2.0
Einführung, Programme und Referenz, dpunkt 1997

[2] Jed Hartman, Josie Wernecke: The VRML 2.0 Handbook
Building Moving Worlds on the Web, Addison−Wesley 1997

[3] Rolf Däßler, Hartmut Palm: Virtuelle Informationsräume mit VRML
Informationen recherchieren und präsentieren in 3D, dpunkt 1998

[4] div. Autoren: Late Night VRML 2.0 with Java
−Unleash the power of VRML and Java, ZDPress 1997

[5] Andreas Steinbruch: VRML, Möglichkeiten und Perspektiven
Diplomarbeit 1996, http://www.iib.bauing.tu−darmstadt.de/~erlacher/doku−mentationen/vrml/vrml/vrml.htm

[6] VNet−Homepage mit Beispielwelten
http://www.csclub.uwaterloo.ca/~sfwhite/vnet/

[7] Deep Matrix.Homepage
http://vienna.eas.asu.edu/~mercator/matrix.html

[8] Dreidimensionale Begegnungen
ZDnet Deutschland, Ausgabe 8/98
http://www.zdnet.de/produkte/artikel/sw/199808/3d06−wf.html

[9] Late Night VRML with Java (Webseite zum gleichnamigenen Buch)
http://ece.uwaterloo.ca/~broehl/vrml/lnvj/index.html

[10] Web3D Consortium: VRML97 Specification, ISO/IEC 14772−1:1997
http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm

[11] Web3D Consortium:VRML200x Specification, ISO/IEC 14772 (Draft)
http://www.web3d.org/TaskGroups/x3d/specification/index.html

[12] David Chatterton, Java und VRML, Silicon Graphics Inc.
http://reality.sgi.com/hendin/vrml+java/htdocs/javaus/javaus.html

[13] „Living Worlds“, Specification Code and Comments’
http://www2.blaxxun.com:1080/livingworlds/spec/draft_1/lw_code.htm

[14] Bill Yerazunis, Barry Perlman: „Open Community“,
High Level Overview of
http://www.meitca.com/opencom/ov.html

[15] Chris Marrins: Anatomie of a VRML Browser
http://www.marrin.com/vrml/Interface.html

[16] Web3D Consortium
http://www.vrml.org

[17] Living World Proposal, Draft 2
http://www.vrml.org/WorkingGroups/living−worlds/draft_2/index.htm

Foren und Newsgoups

comp.languages.vrml
Englischsprachige Newsgroup

http://www.vrml−fokus.de/
Deutschsprachiges VRML Forum

http://web3d.about.com/compute/web3d/
Focus on Web3D− Englischsprachiges Archiv von Artikeln über VRML, enthält zusätzlich ein umfangreiche Sammlung von Links

Online Tutorials

Dave Nadeau, John Moreland, Mike Heck: Introduction to VRML 97
http://www.sdsc.edu/~nadeau/Courses/Siggraph98vrml/

Bob Crispen: VRML Works
http://fly.hiwaay.net/~crispen/vrmlworks/

Firmen

Blaxxun Community Inc.
http://www.blaxxun.com/

Geometrek
http://www.geometrek.com

Microsoft's VRML Project
http://www.microsoft.com/vrml/content.htm

Parallel Graphics
http://www.parallelgraphics.com

Platinum Software
http://www.platinum.com

Sony's Community−Place
http://www.community−place.com

11 Anhang

Anhang A: Das Paket vrml.external

```
vrml.external
|
+- vrml.external.Browser
+- vrml.external.Node
+- vrml.external.field
|   +- vrml.external.field.EventIn
|   |   +- vrml.external.field.EventInMFColor
|   |   +- vrml.external.field.EventInMFFloat
|   |   +- vrml.external.field.EventInMFInt32
|   |   +- vrml.external.field.EventInMFNode
|   |   +- vrml.external.field.EventInMFRotation
|   |   +- vrml.external.field.EventInMFString
|   |   +- vrml.external.field.EventInMFVec2f
|   |   +- vrml.external.field.EventInMFVec3f
|   |   +- vrml.external.field.EventInSFBool
|   |   +- vrml.external.field.EventInSFColor
|   |   +- vrml.external.field.EventInSFFloat
|   |   +- vrml.external.field.EventInSFImage
|   |   +- vrml.external.field.EventInSFInt32
|   |   +- vrml.external.field.EventInSFNode
|   |   +- vrml.external.field.EventInSFRotation
|   |   +- vrml.external.field.EventInSFString
|   |   +- vrml.external.field.EventInSFTime
|   |   +- vrml.external.field.EventInSFVec2f
|   |   +- vrml.external.field.EventInSFVec3f
|   |
|   +- vrml.external.field.EventOut
|   |   +- vrml.external.field.EventOutMField
|   |   |   +- vrml.external.field.EventOutMFColor
|   |   |   +- vrml.external.field.EventOutMFFloat
|   |   |   +- vrml.external.field.EventOutMFInt32
|   |   |   +- vrml.external.field.EventOutMFNode
|   |   |   +- vrml.external.field.EventOutMFRotation
|   |   |   +- vrml.external.field.EventOutMFString
|   |   |   +- vrml.external.field.EventOutMFVec2f
|   |   |   +- vrml.external.field.EventOutMFVec3f
|   |   |
|   |   +- vrml.external.field.EventOutSFBool
|   |   +- vrml.external.field.EventOutSFColor
|   |   +- vrml.external.field.EventOutSFFloat
|   |   +- vrml.external.field.EventOutSFImage
|   |   +- vrml.external.field.EventOutSFInt32
|   |   +- vrml.external.field.EventOutSFNode
|   |   +- vrml.external.field.EventOutSFRotation
|   |   +- vrml.external.field.EventOutSFString
|   |   +- vrml.external.field.EventOutSFTime
|   |   +- vrml.external.field.EventOutSFVec2f
|   |   +- vrml.external.field.EventOutSFVec3f
|   |
|   +- vrml.external.field.EventOutObserver
|   +- vrml.external.field.FieldTypes
|
+- vrml.external.exception
+- vrml.external.exception.InvalidEventInException
+- vrml.external.exception.InvalidEventOutException
+- vrml.external.exception.InvalidNodeException
+- vrml.external.exception.InvalidRouteException
+- vrml.external.exception.InvalidVrmlException
```

Anhang B: VNET als Grundlage für ein erweitertes virtuelles 3D Chatsystem

VNET ist ein von Stephen White and Jeff Sonstein geschriebenes VR System welches nur die offenen Standards von VRML 2.0 und Java verwendet. Es ist frei verfügbar und wird unter der GPL entwickelt und vertrieben. Die im folgenden gemachten Ausführungen stellen teilweise eine Erweiterung der offiziellen Dokumentation dar, beschränken sich der Übersichtlichkeit halber aber nur auf die Teilbereiche welche für die Modifikation der virtuellen Welt von Bedeutung sind. Die originale Dokumentation selber enthält lediglich eine einfache Installationsanweisungen sowie eine Kompatibilitätsliste (siehe Anhang), der frei verfügbare Quellcode ist bis auf auf die Hinweise zur GPL nirgendwo dokumentiert. Weiterhin befindet sich im WWW noch eine Seite über das bei VNET verwendete VIP−Protokoll, auf welches im folgenden Abschnitt eingegangen wird.

Protokoll[24]

Das bei VNET eingesetzte VRML Interchange Protokoll (VIP) ist ein einfaches Protokoll zur Übertragung von VRML−Feldern über Netzwerke um eine Interaktion verschiedener VRML−Welten zu ermöglichen. Es ist auf TCP/IP und so gestaltet das es das selbe Format wie die Java Klassen *java.io.DataInputStream* und *DataOutputStream* benutzt. VIP kodiert jedes der 19 Datentypen in VRML in einem eigenen Feld. Das erste Byte ist ein Tag der angibt um welchen Datentypen es sich handelt, die darauffolgenden Bytes enthalten den entsprechenden Wert der wie folgt kodiert wird:

Tag	**Typ**	**Encoding**	**Beschreibung**
−1	(none)		Keine Daten
0	SFBool	byte	Zero = FALSE, non−zero = TRUE
1	SFColor	float float float	r, g, b
2	SFFloat	float	
3	SFImage	int int int [int int int ...]	width, height, depth, [pixels...]
4	SFInt32	int	
5	SFNode	int	Object id of the root node.
6	SFRotation	float float float float	3−vector of axis, angle
7	SFString	utf8	
8	SFTime	double	Seconds since the "epoch" jan 1 1970 GMT
9	SFVec2f	float float	x, y
10	SFVec3f	float float float	x, y, z

24 www.csclub.uwaterloo.ca/%7Esfwhite/vnet/VIP.html

Tag	Typ	Encoding	Beschreibung
11	MFCo−lor	int [float float float ...]	n, followed by n (r, g, b) 3−tuples
12	MFFloat	int [float float float ...]	n, followed by n floats
13	MFInt32	int [int int int ...]	n, followed by n ints
14	MFNode	int [int int int ...]	n, followed by n object id’s
15	MFRota−tion	int [float float float float, ...]	n, followed by n SFRotation 4−tuples
16	MFString	int [utf8 utf8 utf8 ...]	n, followed by n utf8−encoded strings
17	MFVec2f	int [float float, ...]	n, followed by n (x, y) 2−tuples
18	MFVec3f	int [float float float, ...]	n, followed by n (x, y, z) 3−tuples

Server

Der VNET Server ist wie auch der Client vollständig in Java implementiert wor−den. Durch die Sicherheitsbestimmungen innerhalb einer verteilten Java−Umge−bung muß der das Programm auf dem selben Server gestarted werden von wel−chem auch das VNET−Client−Applet durch den HTML−Browser geladen wird. Um den Server zu starten wird folgender Befehl in einer Shell eingegeben:

```
java VSystem 8888
```

8888 ist der Port auf welchem der Server senden und empfangen soll. Prinzipiell kann hier jeder andere verfügbare Port angegeben werden, in diesem Fall muß dann selbstverständlich auch das VNET−Empfangsapplet auf diesen Port einge−stellt werden (siehe Beschreibung des Client) Von diesem Programm werden dann alle weiteren Klassen initialisiert und aufgerufen:

- Benutzerverwaltung: VUser, VObject
- VRML−Datentypen: VField, VSFBool, VSFRotation, VSFString, VSFVec3f
- Sockets: VFieldInputStream, VFieldOutputStream, VIP
- Nachrichtenverwaltung: Message, MessageQueue
- Sender: WriterThread, WriterThreadObserver

In der Klasse VIP ist das im vorherigen Abschnitt beschriebene VRML−Inter−change−Protokoll implementiert. Für jede eintreffende Verbindungsanforderung wird ein eigener Thread eröffnet und eine MessageQueue angelegt. Bei Nachrich−

ten wird zwischen einfachen Strings und VIP−Daten unterschieden. Normale Strings sind Ausgaben des Chat−Applets und werden entweder an alle ange−schlossenen Clients oder nur an einen/mehrere bestimmte Clients gesendet wenn dieses der Sender so vorgesehen hat. Eine VIP Nachricht repräsentiert einen Avatar und enthält neben dem Namen des Absenders dessen aktuellen Standort, angegeben als XYZ−Koordinaten, Neigung/Drehung und Blickrichtung. VIP−Daten werden stets an alle angeschlossenen Clients, mit Ausnahme des Senders, verteilt.

Client

Der Client kann wahlweise in einem eigenen Applet Browser Fenster laufen oder als eingebettetes Applet auf der selben HTML−Seite auf welcher sich auch der VRML Browser befindet. Es stehen eine ganze Reihe von Parametern zur Verfü−gung um das Aussehen des Chatinterfaces bzw. die Arbeitsweise des Clients zu beeinflussen:

Legt die Farbe des Hintergrundes fest:
```
<PARAM NAME="BGCOLOR" VALUE="A0A0FF">
```

Legt die Farbe des Vordergrundes fest:
```
<PARAM NAME="FGCOLOR" VALUE="000000">
```

Bestimmt den Port auf dem der Client sendet/empfängt
```
<PARAM name="PORT" value="8888">
```

Namen der verfügbaren Avatare
```
<PARAM name="AVATAR_NAMES" value="TinMan, Angel, HalfMoon,
Custom">
```

URLs der verfügbaren Avatare
```
<PARAM name="AVATAR_URLS"
value="TinMan.wrl,Angel.wrl,halfmoon.wrl,">
```

Avatare

Dieser Abschnitt beschreibt die Einbindung von Avataren in eine VR Umgebung die von VNET unterstützt wird. Prinzipiell ist es möglich jeden Avatar der auf einen öffentlich zugänglichen Server liegt einzubinden, dabei sollten aber einige Dinge beachtet werden:

- Der Avatar sollte eine angemessene Größe haben, angemessen bedeutet hier

ein Maximum von 100kByte für die Geometrie und alle Texturen

- Der Avatar muß zentriert sein
- Die Blickrichtung des Avatars soll in Richtung der +Z Achse verlaufen, d.h. beim Laden des einzelnen Avatars in einen VRML−Browser sieht man direkt in dessen Gesicht.
- Der Avatar sollte im Verhältnis nicht die Größe der virtuellen Welt über−schreiten, ein übliches Maß ist 0.25m Breite, 1.6m Höhe and 0.75m Tiefe.
- Wenn möglich sollten keine Lichter/Lichteffekte innerhalb des Avatars einge−bunden werden. Obwohl dies keine eigentliche programmtechnische Ein−schränkung ist (und in den meisten Fällen funktioniert) so besitzen jedoch einige Browser Einschränkungen welche die Anzahl der verwendbaren Licht−quellen beschränken, außerdem stellen einen große Anzahl von Lichtquellen erweiterte Anforderungen and die Geschwindigkeit der Hardware so das die Performanz sehr schnell sinkt.
- Der Avatar selber muß auf einen für alle Benutzer zugänglichen Server abge−legt werden.

Animationen

Zusätzlich zur Einbindung von Avataren in die virtuelle Welt stellt VNET noch eine Möglichkeit zur Verfügung einfache Animationen an den Avataren durch den Eigentümer vornehmen zu lassen (Gesichtsbewegungen, Armbewegungen usw.) Obwohl im vorliegenden Projekt von dieser Möglichkeit kein Gebrauch gemacht wird soll jedoch hier kurz auf die Implementation von solchen Anima−tionen eingegangen werden. Um einem Avatar dahingehend zu erweitern muß zunächst der Avatar als Prototype deklariert werden und anschließend einige zusätzliche Felder in den Quellcode eingefügt werden:

```
    PROTO MyAvatar [
exposedField  MFString    gestures [ "laugh" "smile"
"frown" ]

        eventIn       SFBool      laugh
        eventIn       SFBool      smile
        eventIn       SFBool      frown
        ] {

### avatar geometry goes here
}

MyAvatar {}
```

Nach dem Laden des Avatar−File sucht VNET im ersten Node nach einem *MFString* mit der Bezeichnung *gestures.* Findet es diesen werden die dazugehöri−gen *EventIns* Felder eingebunden sowie entsprechende Java−Buttons in das Chat−Applet integriert. Beim Drücken eines solchen Buttons wird ein *set_value* auf das entsprechende Feld sowohl lokal als auch in allen angeschlossenen Welten ausge−

löst. Die entsprechende Konvertierung des *EventInSFBool* nach *SFTime* und schließlich zur gewünschten Animation ist als JavaScript im Quellcode des Avatars durchzuführen.

Anhang C: Standardelemente in der VRML Welt

Wand

Abmessungen (BxHxT): 5,0m 2,9m 0,01m
Textur: keine
Farbe: diffuseColor 0.75 0.75 0.7
shininess 0.2
Dateiname: wand.wrl

Tür

Abmessungen (BxHxT): 0,86 2,0 0,04m
Textur: wood.jpg
Farbe: keine
Dateiname: door.wrl

Computertische

Abmessungen (BxHxT): 1,2m 0,75m 0,8m
Textur: keine
Farbe: diffuseColor 0.75 0.75 0.7
shininess 0.2
Dateiname: table.wrl

Büroraum

Abmessungen (BxHxT): 3,56 2,9 7,0m
Dateiname: smallroom.wrl

Diese Angaben stellen nur einen Richtwert für ein normales Büro in der 5. Etage dar und können bei größeren/kleineren Räumen entsprechend verändert werden. Einzig die Höhe von 2,9 Metern ist bindend. Ein Standardraum entsteht durch Verwendung von zwei Inline−Nodes *wall.wrl*, einem Inline−Node *windowwall.wrl* und einen Inline−Node *doorwall.wrl*. Die beiden Wände müssen zusätzlich um den Faktor 1,4 skaliert und um 90 Grad gedreht werden:

```
#Rechte Wand
   Transform {
            translation 1.78 1.45 −3.5
               rotation 0.0 1.0 0.0 1.57
                              scale 1.4 1.0 1.0
            children Inline { url "wand.wrl" }
        }
```

Avatare

Abmessungen: variabel. Da Avatare als externe Dateien durch das VNET−Applet in die Welt eingefügt werden ist darauf zu achten, das diese auf ein entsprechen−des Maß zu skalieren sind. Richtwert ist eine normale Körpergröße von 1,60 bis 1,90m. **Wichtig:** VNET selbst prüft nicht nach ob ein Avatar zu groß ist!

Anhang D: Quellkode des Objektservers

```
//ObjectServer.java
import java.net.*;
import java.io.*;
import java.util.*;
public class ObjectServer implements Runnable
{
    public static final int PORT = 8765;
    protected ServerSocket listen;
    protected Vector connections;
    Thread    connect;
    ObjectDB db = new ObjectDB();
    static FileWriter log;
    static boolean logging = false;

            public ObjectServer()
            {
              System.out.println ("Starting objectserver");
               try
               {
                       listen = new ServerSocket(PORT);
               } catch (IOException e)
               {
                       System.err.println("Error! Couldn't create sockets:"+e);
                       System.exit(1);
               }
               connections = new Vector();
               connect = new Thread(this);
               connect.start();
               System.out.println("Objectserver is running and listening on Port
8765");
            }

            public void run()
            {
               try

               {
                       while(true)
```

```
                        {
                                Socket client=listen.accept();
                                connection c = new connection(this, client);
                                connections.addElement(c);
                                System.out.println("New connection accepted from "
                                        + client.getInetAddress().getHostName());

                                if (db.length() >=0) {
                                for (int z=0;z<db.length();z++)
                                  c.out.println(db.getEntry(z));
                                }
                        }
                   } catch (IOException e)
                   {
                           System.err.println("Error while waiting for connection:"+e);
                           System.exit(1);
                   }
              }
              public static void main(String[] args)
              {
                       if (args.length == 1) {
                      try {
                          log = new FileWriter(args[0]);
                          logging = true;
                      } catch (IOException e) {
                          System.out.println("Error while creating logfile");
                      }
                      if (logging) System.out.println ("Logging to file "+ args[0]+"
enabled");
                   }
                   new ObjectServer();
              }
              public void broadcast(String msg)
              {
                   int i;
                   connection you;

                   for (i=0; i<connections.size(); i++)
                   {
                           you = (connection) connections.elementAt(i);
                           you.out.println(msg);
```

```
                }
                System.out.println ("Got message and passed away: "+msg+" ");
                try
                    {
                        if (logging) log.write(msg);
                    } catch (IOException e)
                        {
                            System.err.println("Error! Cannot write to logfile");}
                if (db.exists(msg)>=0) {
                  db.remove((db.exists(msg)));
                  System.out.println ("Removed "+msg);}
                else {
                  db.addEntry(msg);
                  System.out.println ("Added "+msg);}
                db.listAll();
            }
}
```

```
//connection.java
import java.net.*;
import java.io.*;

class connection extends Thread
{
            protected Socket client;
            protected DataInputStream in;
            protected PrintStream out;
            protected ObjectServer server;

            public connection(ObjectServer server, Socket client)
            {
               this.server=server;
               this.client=client;
               try
               {
                       in = new DataInputStream(client.getInputStream());
                       out = new PrintStream(client.getOutputStream());
               } catch (IOException e)
               {
                       try { client.close(); } catch (IOException e2) {} ;
                       System.err.println("Error −  Cannot create streams: " + e);
                       return;
               }
               this.start();
            }

            public void run()
            {
               String line;
               try
               {
                       while(true)
                       {
                               line=in.readLine();
                               if(line!=null)
                                       server.broadcast(line);
                       }
```

```
            } catch (IOException e)
            {
                    System.out.println("Error:" + e);
            }
        }
}
```

```
//objectDB

import java.util.Vector;
import java.util.Enumeration;

public class ObjectDB {
  Vector db   = new Vector();
  void addEntry (String element) {
   db.addElement(element);
  }
  int exists (String element) throws ArrayIndexOutOfBoundsException {
    if (db.isEmpty()) return(-1);
    else
      for (int z=0; z<db.size();z++) {
            if (element.equals(db.elementAt(z))) return (z);
      }
    return (-1);
  }
  int length() {
    return (db.size());
      }


  String getEntry(int x) throws ArrayIndexOutOfBoundsException {
    if (x>=0 && x< db.size()) return(db.elementAt(x).toString());
    else if (db.isEmpty()) System.out.println("Error! Database is empty");
         else  System.out.println("Error! Database is smaller!");
    return("");
      }

  void remove(int x) {
    if (x>=db.size())  System.out.println("Error! Database is smaller!");
     else db.removeElementAt(x);
  }

  void listAll() {
    for (Enumeration el=db.elements(); el.hasMoreElements(); )
      {
            System.out.println((String)el.nextElement());
      }
```

```
  }
}
```

Anhang E: Quellcode des Objektclient−Applets

```
// protoParser.java

import java.applet.*;
import java.util.*;
import java.awt.*;
import java.io.*;
import java.net.*;

import vrml.external.field.*;
import vrml.external.*;
import vrml.external.Browser;
import vrml.external.Node;
import vrml.external.exception.*;
import vrml.field.*;
import vrml.node.*;


public class protoParser  extends Applet implements EventOutObserver {

         Browser     browser;          // the VRML browser
            Node     objectNode;
         boolean     local;
          String[]   objectNames;
  VRMLobjectType[]   objects; // my own Datatype to handle Events from VRML
  VRMLobjectClient   client;
  //  TextField output = null;


    public String getAppletInfo() {
        return "VRML Object Interface 0.2 by Andreas Mueller";
    }



  public void init() {
    System.out.println("protoParser.init()...");

  }
```

```
public void start() {

EventOutMFString   objectList;
                int   no;
            URL   urlbase = this.getCodeBase() ;

System.out.println("Applet.start()...");


//    output = new TextField(40);
//    add(output);


Browser tmp_browser = null;
for (int i = 0; i < 10; i++)
  {
        try
          {
            System.out.println("Getting the browser....");

            tmp_browser = (Browser)Browser.getBrowser( this, "", 0);
            System.out.println("Getting the browser....1");
          }
        catch ( NullPointerException npe )
          {
            System.out.println( "NullPointerException getting browser"
                        );
          }
        if ( tmp_browser != null ) { break; }
            System.out.println("Getting the browser....2");
        try { Thread.sleep ( 500 ); }
        catch ( InterruptedException ie ) { }
        System.out.println( "null browser, retry " + i );
  }
            System.out.println("Getting the browser....3");
if ( tmp_browser == null )  { throw new Error ("getBrowser failed"); }
else if ( tmp_browser != browser) // detect if VRML plugin reloaded
  {
        browser = tmp_browser;
```

```
      }

    System.out.println("Got the_ browser: " + browser);

    System.out.println("Parsing nodes...");

    try {
      objectNode = browser.getNode("ObjectList");
      System.out.println("Got ObjectList node....");
    }
    catch (InvalidNodeException e) {System.out.println("Error!! Couldn’t get
ObjectList node ");
    }

    try { objectList = (EventOutMFString) objectNode.getEventOut("objects");
    System.out.println("Got ObjectList string....size"+objectList.getSize());
    objectNames = objectList.getValue();

    if (objectList != null) {
      no = objectList.getSize();
      objects = new VRMLobjectType[no];
      for (int i = 0; i < no; i++)
            {
              System.out.println("Getting node "+i+": "+objectNames[i]);
              objects[i] = new VRMLobjectType(objectNames[i], browser);
              objects[i].restart(browser, this);


            }
    }

    } catch (InvalidEventOutException e) {
      System.out.println("Error! No objects found in ObjectList");
    }

    client= new VRMLobjectClient();  //  the client that connects to the outside
world
                 if (client.connect(urlbase, objects, getFrame()))
                   {
                  local=false;
                   }
```

```
                else
                  {
                local=true;
                  }
    }

  public void stop() {
    System.out.println("stopping... ");
    client.stop();
    browser = null;
    for (int i = 0; i < objects.length; i++) {
            objects[i].stop();
    }

    Runtime.getRuntime().gc();  //Calling the garbage collector
  }

  public void destroy() {
    System.out.println("preparing for unloading...");
    System.out.println("stopping... ");
    client.stop();
    browser = null;
    for (int i = 0; i < objects.length; i++) {
            objects[i].stop();
    }

    Runtime.getRuntime().gc();  //Calling the garbage collector

  }



  public void callback(EventOut event, double time, Object userData) {

    EventOutSFBool   active = (EventOutSFBool) event;
    VRMLobjectType   who = (VRMLobjectType) userData;

    if (active.getValue()) {
    System.out.println("ObjectType: EAI callback()....from: "+who.getName()+"
node");
```

```
    System.out.println("Click!!");

    if (local) {
      who.set_status();
    }
    else {
      client.SendNode(who.getName());
    }
    }
    }

  private Frame getFrame() {
    Container c = getParent();
    while (c != null && !(c instanceof Frame)) {
      c = c.getParent();
    }
    return (Frame)c;
  }
}
```

```
//VRMLobjectClient.java

import java.io.*;
import java.net.*;
import java.util.*;
import java.awt.*;
import java.applet.*;
import java.lang.String;

import vrml.external.field.*;
import vrml.external.*;
import vrml.external.Browser;
import vrml.external.Node;
import vrml.external.exception.*;
import vrml.field.*;
import vrml.node.*;

/* VRMLobjectClient */


public class VRMLobjectClient implements Runnable
{
  Socket             socket;
  DataInputStream    in;
  PrintStream        out;
  Thread             thread;
  VRMLobjectType[]   objects;
  boolean            isConnected;

  boolean connect(URL url, VRMLobjectType[]  objects, Frame frame) {

    this.objects = objects;

    boolean done=false;
    isConnected=true;

    while (done != true)
      {
            done = true;
            try
```

```
              {
                socket = new Socket(url.getHost(), 8765);
                System.out.println("Connecting to Objectserver.....");
                in = new DataInputStream(socket.getInputStream());
                out = new PrintStream(socket.getOutputStream());
              } catch (IOException e)
                {
                  isConnected=false;
                  connectionRequester cr = new connectionRequester(frame, true);
                  cr.show();

                  if(cr.getResult()==false) done=true;
                  else done=false;

                  System.out.println("Verbindung zum Server
fehlgeschlagen!"+done);
                }
            System.out.println(done);
      }


    if (isConnected)
      {
            System.out.println("Verbindung zum Server aufgenommen...");

    if (thread == null)
      {
            thread = new Thread(this);
            thread.setPriority(Thread.MIN_PRIORITY);
            thread.start();
      }
      }
    return (isConnected);
}



  void SendNode(String nodeName) {

        //Eingabezeile an ECHO-Server schicken
```

```
  try {
    System.out.println("Send Node to server...");
    out.write(nodeName.getBytes());
    out.write('\n');
  } catch (IOException e) {System.out.println("SendNode failed!");}
  System.out.println("Sent Node completed");
}

public void stop()
{
  try
    {
          socket.close();
    } catch (IOException e)
          {  }

  if ((thread !=null) && thread.isAlive())
    {
          thread.stop();
          thread = null;
    }
}

public void run()
{
  String line;

  try
    {
          while(true)
            {
             line = in.readLine();
             if(line!=null)
               {
            System.out.println(line);

            for (int z=0; z< objects.length; z++) {
              if ( line.equals(objects[z].getName())) {
                System.out.println(z+" matches");
                  objects[z].set_status();
```

```
                    if (objects[z].getURL().length() > 4)
                                objects[z].loadURL();
                  }
                 }
                    }
                }
      } catch (IOException e) { System.out.println("Verbindung zum Server abgebro−
chen"); }
  }
}
```

```
// VRMLobjectType.java
import java.util.*;
import java.awt.*;
import java.io.*;
import java.net.*;
import java.applet.*;

import vrml.external.field.*;
import vrml.external.*;
import vrml.external.Browser;
import vrml.external.Node;
import vrml.external.exception.*;
import vrml.field.*;
import vrml.node.*;



public class VRMLobjectType{
  String           name;
  String[]         url = new String[1];
  Node             node;
  boolean          status = false;
  Browser          browser;
  String[]         frame = new String[1];

  EventOutSFBool   isActive;
  EventInSFBool    setStatus;
  EventOutSFBool   status_changed;
  EventOutSFString urlstring;
  EventOutSFString framestring;

  public VRMLobjectType(String name, Browser browser) {
    this.name = name;
    this.browser = browser;

  }

  public String   getName() { return name; }  //returns the name of the VRML-
object
  public String   getURL() { return url[0]; } //returns the URL
  public void     loadURL() {
```

```
    browser.loadURL(url, frame );
    System.out.println("Load URL: "+url+"in Frame: "+frame);
  }

  public void set_status() {    //toggles the value of status
    setStatus.setValue(true);
  }

  public void restart(Browser browser, EventOutObserver observer) {
    node = null;

    while (node == null) {
      try { node = browser.getNode(name); }
      catch (InvalidNodeException e) {System.out.println("Error!! Couldn’t get
node: "+name); }
    }
    isActive = (EventOutSFBool) node.getEventOut("isActive");
    isActive.advise(observer, this);
    System.out.println("Got eventOut: isActive");
    setStatus = (EventInSFBool) node.getEventIn("set_status");
    System.out.println("Got eventIn: set_status");

    urlstring = (EventOutSFString) node.getEventOut("URL");
    url[0] = urlstring.getValue();
    System.out.println("Got eventOut: URL, length: " +
urlstring.getValue().length());

    framestring = (EventOutSFString) node.getEventOut("frame");
    if (framestring.getValue().length()>0)
      frame[0] = "target="+framestring.getValue();
    else frame[0]="";
    System.out.println("Got eventOut: frame");

  }

  public void stop() {
              node = null;
         isActive = null;
    status_changed = null;
         setStatus = null;
         urlstring = null;
```

```
        framestring = null;

    }
}
```

```
// connectionRequester.java

import java.awt.*;
import java.awt.event.*;

public class connectionRequester
extends Dialog
{
  private  boolean result;
  private  Button _retry, _cancel;

  public connectionRequester(Frame f, boolean modal)
  {
      super(f, modal);

      //Fenster

      setLayout(new FlowLayout());
      resize (270,60);
      setBackground(Color.lightGray);
      setTitle("Fehler! Keine Verbindung zum Server");

      _retry = new Button("Nochmal versuchen?");
      _cancel = new Button("Lokal arbeiten");

      add(_retry);
      add(_cancel);

      setResizable(false);
      setLocation(100, 100);

      //       pack();
   }


   public boolean action(Event event, Object object)
   {
     if (event.target == _retry || event.target == _cancel) {
           if (event.target == _retry) {
             result=true;
```

```
                } else {
                  result = false;
                }
             hide();
         dispose();
         return true;
        }
        return false;
     }
    public boolean getResult()
    {
      return result;
    }
}
```