Vorlesung im
Sommersemester 1998
Dr. Stephan Diehl
Universität des Saarlandes

Implementierung verteilter, virtueller Welten im Internet

Vertraulich, bitte nicht weiterverteilen!

Leider sind einige Kapitel erst stichwortartig dargestellt.

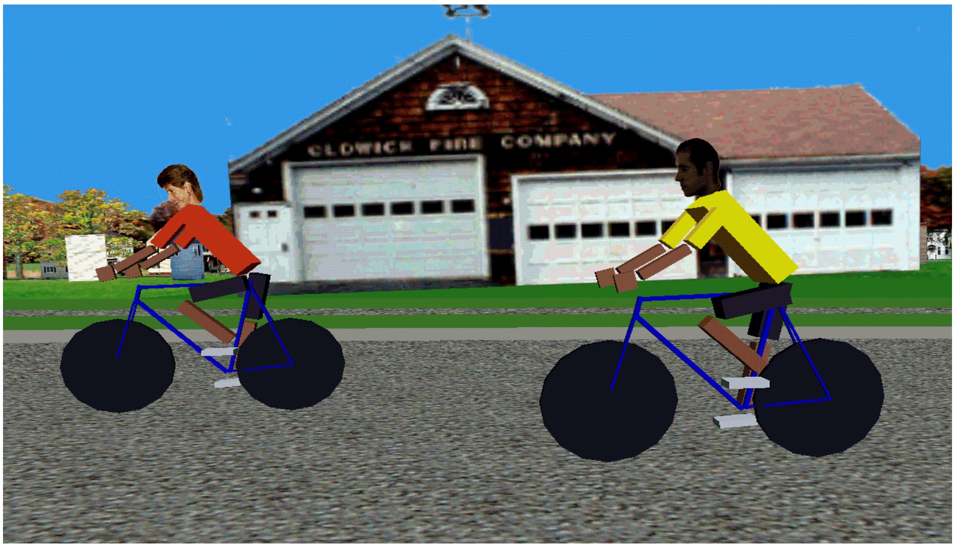
Inhaltsverzeichnis

Einleitung

Ein Beispiel: Peloton

Im Februar 1998 wurde auf der VRML98, der Konferenz zum Thema Virtual Reality Modeling Language, in Kalifornien ein System namens Peloton vorgestellt. Der Begriff "Peloton" stammt aus dem Radsport und bezeichnet das Hauptfeld in einem Radrennen. Peloton wurde in den Bell Laboratories, einer Forschungseinrichtung von Lucent Technologies, entwickelt. Peloton ist eine verteilte Radfahrsimulation, ein computerbasiertes System zum Erzeugen virtueller Umgebungen für simulierte Fahrradfahrten.

Mehrere Personen, die an unterschiedlichen Rechnern im Internet angeschlossen sind, können gleichzeitig durch virtuelle Landschaft radeln. Sie können einfach nur die Landschaft genießen, aber auch Rennen fahren. Um an einem solchen Rennen teilzunehmen, melden sich die Fahrer über das World Wide Web am Peloton-Server an. Die Teilnehmer verfügen über stationäre Fahrräder (Trimmräder), die an ihren lokalen Computer über Sensoren und sogenannte Gegenkraftsteuerungen ("force-feedback devices") angeschlossen. Letztere sorgen dafür, daß der Pedalwiderstand vom Computer gesteuert werden kann. Für den Radler bedeutet dies eine neues Multimedia-Erlebnis: er muß sich wirklich anstrengen, wenn er einen Berg hochfahren will. Die Sensoren liefern Informationen, die zum Positionieren des Radlers in der virtuellen Umgebung verwendet werden. Die virtuelle Umgebung basiert auf topologischen Daten tatsächlicher Radstrecken und besteht aus computergenerierten Grafiken, sowie Bildern und Geräuschen der Radstrecke. In dieser virtuellen Umgebung wird der Radler als animiertes grafisches Objekt dargestellt. Solch ein einen Benutzer repräsentierendes Objekt wird als Avatar bezeichnet. In unserem Beispiel bewegt er sich so schnell, wie der Radler in die Pedale tritt. Jeder Radler hat einen eigenen Blick auf die Radstrecke, und kann seine Perspektive auch während der Fahrt ändern. Außerdem kann er sich per "webphone" oder auch mit einem konventionellen Telefon mit anderen Radlern unterhalten.



Da Peleton mit herkömmlichen WWW-Werkzeugen entwickelt wurde und verbreitete Datenformate verwendet, läuft es auf vielen Systemen. Darüber hinaus paßt es die

Komplexität der Darstellung an die Fähigkeiten des jeweiligen Rechners an. Abschließend kann man sagen, daß Peloton eine neue Klasse von Anwendungen repräsentiert, die es Benutzern erlaubt, in Echtzeit mit Hilfe von unkonventionellen Ein-/Ausgabegeräten zu interagieren. Die Entwickler von Peloton wollen das System nun so erweitern, daß Jogger auf Laufbändern und Ruderer in Trimmgeräten zusammen mit Radlern dieselbe Umgebung erleben können.

Architektur von Peloton

Schaut man sich die Architektur von Peloton genauer an und vergleicht sie mit anderen Mehrbenutzerwelten, die derzeit für´s Internet entwickelt werden, dann stellt man fest, daß diese ähnliche Implementierungstechniken, Sprachen und Protokolle verwenden. Die dreidimensionalen Objekte sind in VRML, die Programmteile in Java kodiert. VRML und Java kommunizieren über das External Authoring Interface (EAI). Und schließlich werden für die Netzwerkkommunikation HTTP, CORBA und/oder die Internetprotokolle TCP und UDP verwendet.

Ein weiteres Beispiel: Eine virtuelle Theateraufführung

Peloton zeigt aber nur einen Bruchteil der Möglichkeiten , die verteilte, virtuelle Welten eröffnen. Die im Folgenden geschilderte virtuelle Welt ist fiktiv. Allerdings werden heute schon Teile davon im VRMLDream Projekt verwirklicht.

Stellen wir uns vor, ein Theaterstück soll von Schauspielern, die an verschiedenen Rechnern im Netz sitzen und ihre Avatare steuern, aufgeführt werden. Das Bühnenbild wurde zuvor aus mehreren Katalogen verschiedener Anbieter zusammengestellt und online bezahlt. Die Schauspieler suchen sich nun für ihre Rollen passende Avatare und kleiden diese ein. Dabei sind mehrere Schauspieler gleichzeitig in der Garderobe oder auf der Bühne und unterhalten sich über die Wahl der Kleider, ihre Rollen und die schlechte Bezahlung. In der Aufführung soll auch ein “lebender” Elefant auftreten. Er wurde aus einem Katalog ausgewählt und verfügt nun über ein Eigenleben. Die Zuschauer müssen an der Kasse ihre Eintrittskarten vorzeigen, die sie in den letzten Tagen an den Vorverkaufsstellen mit elekronischen Münzen erworben haben. Die Zuschauer können sich miteinander unterhalten, ohne daß es die Aufführung stört, sie können aber auch Zwischenrufe machen. Ein Zuschauer hat eine Zeitbombe mitgebracht. Als Hut getarnt hat er ein “trojanisches Pferd” eingeschleust. Als es während der Vorstellung aktiv wird, reißt es den Schauspielern die Kleider vom Leib. Kurz darauf löscht es alle Objekte dieser Szene auf dem Server.

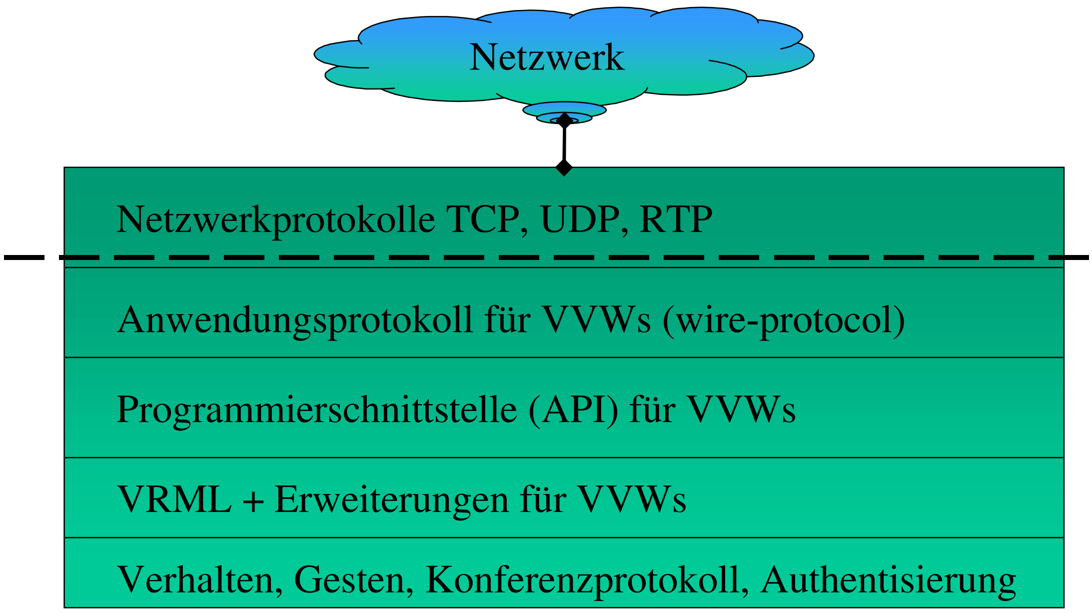
Anforderungen

Aus den obigen Beispielen kann man einige technische Anforderungen an verteilte, virtuelle Welten ersehen. Objekte verfügen über eine grafische Repräsentation, einen internen Zustand, sowie über ein Verhalten (Programmkode) . Solche Objekte sollen in verteilte Szenen in Echtzeit eingefügt bzw. gelöscht, ihr Verhalten in Echtzeit verfolgt und ihre Implementierung auf verschiedene Rechner verteilt werden können. Manche Objekte müssen sich von anderen steuern lassen, und Objekte müssen Informationen austauschen können. Sollen importierte Objekte zu festen Bestandteilen einer Szene werden, muß man sie samt ihres Verhaltens auch abspeichern können. Weitere Anforderungen ergeben sich aus Sicherheitsgründen, etwa dem Schutz vor Vandalismus. Ein Objekt kann den Zugriff auf seine Daten und Funktionen gewähren oder verweigern. Dazu kann es Zugriffsrechte an Benutzer und ihre Objekte vergeben. Daraus ergibt sich allerdings ein weitere Anforderung, nämlich die Identität der Benutzer und ihrer Objekte muß kontrolliert werden. Hierfür verwendet man kryptographische Verfahren. Beim Zugriff auf gemeinsame Resourcen können in verteilten Systemen Konflikte entstehen. Solche Konflikte können durch Transaktionsmechanismen vermieden werden oder durch Konfliktlösungsstrategien nachträglich bereinigt werden.

Anwendungen

Die Anwendungsgebiete verteilter, virtueller Mehrbenutzerwelten sind vielfältig und im Grunde nur durch unsere Fantasie beschränkt. So können sich Planer und Geldgeber schon vor der Grundsteinlegung im virtuellen Pendant eines neuen Gebäude treffen. Ebenso können die Auswirkungen von Baumaßnahmen, etwa neuer Straßen oder Bahntrassen, frühzeitig den Entscheidungsträgern vor Augen geführt werden. In virtuellen Seminarräumen können sich Studenten verschiedener Universitäten zu Kursen treffen (Telelearning). Kooperative Arbeitsvorgänge (z.B. Montage und Demontage von Bauteilen) können ohne Materialverbrauch geübt werden. In Kliniken können mehrere Experten bei komplizierten Operationen zusammenarbeiten (Telemedizin, virtuelles Skalpell). Geräte, Roboter und Anlagen können von mehreren Leuten gesteuert werden, etwa ein Roboter, der Atommüll abfüllen soll (Telepräsenz). Erst am Anfang steht der Einsatz verteilter, virtueller Welten in der Kunst, etwa kollaborative Malereien und Skulpturen.

Auch die Spielindustrie hat die Zeichen der Zeit erkannt. Verfügten noch vor kurzem viele Actionspiele über einen Mehrspielermodus, bei dem die Computer über die serielle Schnittstelle miteinander verbunden wurden, so kommunizieren heute solche Spiele über das Internet.

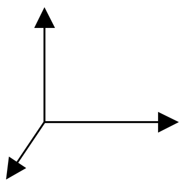


Dreidimensionale Computergrafik

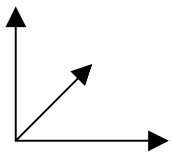
Dreidimensionale Welten bestehen aus dreidimensionalen Objekten, die wiederum aus primitiveren Objekten wie Kugeln, Quadern und Kegeln zusammengesetzt wurden. Beim Zusammensetzen von Objekten können diese transformiert,d.h. z.B. vergrößert oder verkleinert werden. Mathematisch, lassen sich solche Transformationen durch Matrizen beschreiben und die Komposition von Transformationen läßt sich dann durch Multiplikation der zugehörigen Matrizen ausdrücken. Dreh- und Angelpunkt einer VRML-Welt ist das Koordinatensystem. Position und Ausdehnung eines Objektes können in einem lokalen Koordinatensystem definiert werden. Das Objekt kann dann in ein anderes Koordinatensystem plaziert werden, indem man die Position, die Ausrichtung und den Maßstab des lokalen Koordinatensystems des Objektes in dem anderen Koordinatensystem festlegt. Dieses Koordinatensystem und die in ihm enthaltenen Objekte können wiederum in ein anderes Koordinatensystem eingebettet werden.

Einige mathematische Grundlagen

In diesem Abschnitt wird ein wenig auf die mathematischen Grundlagen eingegangen, die hinter dreidimensionalen Computergrafiken stecken. In dreidimensionalen Computergrafiken reduziert man Objekte der realen Welt auf Mengen von Punkten in einem dreidimensionalen Koordinatensystem. In der Regel betrachtet man dabei kartesische Systeme, bei denen die Koordinatenachsen senkrecht aufeinanderstehen. Ein dreidimensionales Koordinatensystem ist entweder rechtshändig oder linkshändig. Diese Bezeichnungen kommen daher, daß, wenn man die Hand geeignet hält, Daumen, Zeigefinger und Mittelfinger der jeweiligen Hand in Richtung der X-, Y- und Z-Achse des Koordinatensystems zeigen.



rechtshändiges Koordinatensystem



linkshändiges Koordinatensystem

Beim Definieren dreidimensionaler Szenen geht man in der Regel so vor, daß man einzelne Objekte in lokalen Koordinatensystemen mit eigenem Ursprung definiert und diese dann durch Rotation, Verschieben, Verkleinern und Vergrößern in ein anderes Koordinatensystem integriert. Betrachten wir einmal den Punkt mit den Koordinaten (X,Y,Z). Für dreidimensionale Grafiken verwendet man gerne sogenannte homogene Koordinaten. Bei diesen gibt man zusätzlich einen Vergrößerungsfaktor w an. Der Zusammenhang zwischen den kartesischen Koordinaten (X,Y,Z) und den homogenen Koordinaten (x,y,z,w) einer Punktes ist folgender:

$$(x,y,z,w) = (X/w, Y/w, Z/w, w) \text{ für } w \text{ ungleich } 0$$

Für w gleich 0 spricht man von Punkten im Unendlichen.
Für w gleich 1 spricht man von homogenisierten Koordinaten.

Im Folgenden wird für homogene Koordinaten nur den Fall der homogenisierte Koordinaten betrachtet. Angenommen, man verschiebt einen Punkt um t_x in X-Richtung, t_y in Y-Richtung und t_z in Z-Richtung. Dann hat der neue Punkt die Koordinaten (x',y',z',w') mit

Für homogene Koordinaten	Für kartesische Koordinaten
$x' = x + t_x$	$x' = x + t_x$
$y' = y + t_y$	$y' = y + t_y$
$z' = z + t_z$	$z' = z + t_z$
$w' = w = 1$	

Das obige Gleichungssystem läßt sich für homogene Koordinaten auch mit Hilfe von Matrizenmultiplikation mit der Verschiebungsmatrix T bzw. für kartesische durch Vektoraddition mit einem Verschiebungsvektor darstellen.

Für homogene Koordinaten

$$(x', y', z', 1) = (x, y, z, 1) * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ t_x & t_y & t_z & 1 \end{bmatrix}$$

Für kartesische Koordinaten

$$(x', y', z') = (x, y, z) + (t_x, t_y, t_z)$$

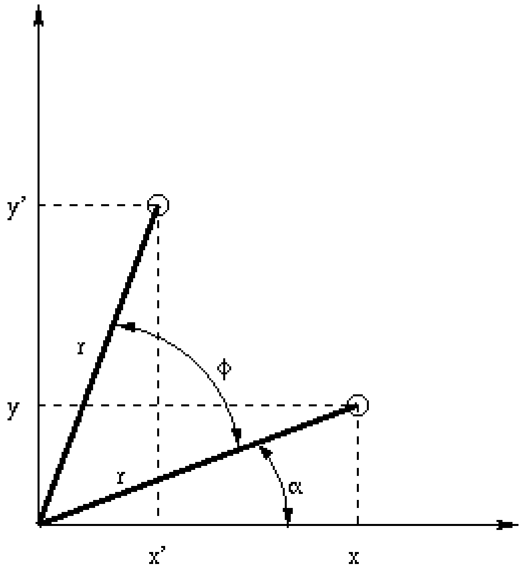
Die Verschiebung kann man rückgängig machen, indem man um -tx ,-ty und -tz verschiebt. Somit erhält man die inverse Matrix T^{-1}:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -t_x & -t_y & -t_z & 1 \end{bmatrix}$$

Wie man leicht nachrechnen kann, ist $T*T^{-1}=I$, wobei I die identische Matrix ist:

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Allerdings ist nicht jede Matrix invertierbar. Nur bei quadratischen Matrizen, deren Determinante ungleich Null ist, ist eine Invertierung möglich. Mit anderen Worten heißt dies, daß es Transformationen gibt, die man nicht rückgängig machen kann.



• Zweidimensionale Drehung um den Ursprung

Betrachtet man nun einmal eine Rotation um Winkel ϕ im Gegenuhrzeigersinn um die Z-Achse, dann stellt man zuerst fest, daß jeder Punkt auf einer zweidimensionalen Ebene rotiert wird, da z'=z. Für die neuen X- und Y-Koordinaten kann man sich daher auf den zweidimensionalen Fall beschränken (siehe Abbildung).

Aus der Definition von Sinus und Kosinus folgt:

$$x = r \cdot \cos(\alpha)$$
$$y = r \cdot \sin(\alpha)$$
$$x' = r \cdot \cos(\alpha+\phi)$$
$$y' = r \cdot \sin(\alpha+\phi)$$

Nun kann man die Sinus- und Kosinusregeln für Summen von Winkeln anwenden:

$$x' = r \cdot \cos(\alpha+\phi) = r \cdot \cos(\alpha) \cdot \cos(\phi) - r \cdot \sin(\alpha) \cdot \sin(\phi)$$
$$y' = r \cdot \sin(\alpha+\phi) = r \cdot \cos(\alpha) \cdot \sin(\phi) + r \cdot \sin(\alpha) \cdot \cos(\phi)$$

Insgesamt erhält man also:

$$x' = x \cdot \cos(\phi) - y \cdot \sin(\phi)$$
$$y' = x \cdot \sin(\phi) + y \cdot \cos(\phi)$$
$$z' = z$$
$$w' = w$$

Auch diese Transformation kann man in Matrizenform schreiben:

Für homogene Koordinaten

$$
(x', y', z', 1) = (x, y, z, 1) * \begin{bmatrix} \cos\phi & \sin\phi & 0 & 0 \\ -\sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

Für kartesische Koordinaten

$$
(x', y', z') = (x, y, z) * \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}
$$

Ebenso kann man andere Rotationen, aber auch Vergrößern und Verkleinern in Achsenrichtung mit einer Matrix beschreiben. Bei kartesischen Koordinaten muß man leider für Verschiebungstransformationen auf Vektoraddition zurückgreifen, bei homogenen Koordinaten lassen sich alle Transformationen durch Matrizen beschreiben. Das Interessante daran ist, daß bei Verwendung von homogenen Koordinaten dem Komponieren von Transformationen das Multiplizieren der zugehörigen Matrizen entspricht. Als Ergebnis erhält man eine einzige Transformationsmatrix. Da die Matrizenmultiplikation nicht kommutativ ist, ist die Ausführungsreihenfolge von Transformationen entscheidend. In einem 3D-Grafiksystem wendet man in der Regel eine Folge von Transformationen auf eine große Menge von Punkten an. Daher berechnet man geschickter Weise zuerst das Produkt der Matrizen und multipliziert dann die resultierende Matrix mit den einzelnen Punkten.

Auch planare Projektionen wie die Parallelprojektion und die perspektivische Projektion, mit denen man ein zweidimensionales Bild der dreidimensionalen Objekte erhält, lassen sich mit Hilfe von Matrizen beschreiben. Es wird dabei eine Punktmenge im dreidimensionalen Koordinatensystem auf eine Ebene im selben Koordinatensystem projiziert.

Zusammenfassend kann man diesen Ansatz, dreidimensionale Computergrafiken zu erstellen, so beschreiben: Eine Menge primitiver dreidimensionaler Objekte, also Punktmengen, wird durch Transformationen, die sich als Produkt von Matrizen darstellen lassen, im Raum plaziert und dann wird durch eine Projektion ein zweidimensionales Bild dieses Raumes erzeugt. Natürlich muß ein 3D-Browser noch viel mehr können. Er muß sichtbare Flächen berechnen, Farbe, Textur und Transparenz von Oberflächen und Schatten, Reflektionen und die Brechung des Lichts berücksichtigen. Unter Umständen muß er auch die Sichtweise bestimmter Kameras, etwa Tiefenschärfe, erzeugen.

Die Grafikpipeline

Viele Grafiksysteme, insbesondere solche, die Grafiken in Echtzeit erzeugen müssen, funktionieren nach dem im Folgenden beschriebenen Prinzip. Objekte sind als mathematische Modelle oder direkt als Polygonnetze gegeben. Polygonnetze definieren die Oberfläche eines Objekts als Menge von Flächen, die Anhand von Eckpunkten festgelegt werden. In beiden Fällen wird die Oberfläche des Objekts in Dreiecke oder in manchen Systemen auch in Polygone aufgeteilt. Diese Dreiecke sind durch die Position ihrer Eckpunkte im lokalen Koordinatensystem des Objekts genau festgelegt. Die Dreiecke durchlaufen nun verschiedene Verarbeitungseinheiten der Anzeigesoftware. Dabei wird die Ausgabe einer Einheit als Eingabe für die nächste verwendet. Man bezeichnet die Folge der Verarbeitungseinheiten auch als Grafikpipeline. Moderne 3D-Grafikkarten bieten Teil- oder gar vollständige Hardwareimplementierungen einiger dieser Verarbeitungseinheiten.

1) Transformation: Das Dreieck wird im globalen Koordinatensystem plaziert, d.h. Drehungen, Verschiebungen und Skalierungen werden angewandt.

2) Backface Culling: Bei den Dreiecken unterscheidet man Rück- und Vorderseiten. Rückseiten müssen in manchen Fällen nicht angezeigt werden. Für solche Dreiecke wird getestet, ob der Betrachter die Rück- oder Vorderseite des Dreiecks sieht. Im Falle der Rückseite wird das Dreieck nicht an die nachfolgenden Verarbeitungseinheiten weitergegeben. Ein Beispiel macht dies klarer: Man kann sich einen Quader wie einen geschlossenen Pappkarton vorstellen. Die Rückseiten der Pappwände kann man von Außen nicht sehen, da sie immer durch andere Wände verdeckt werden.

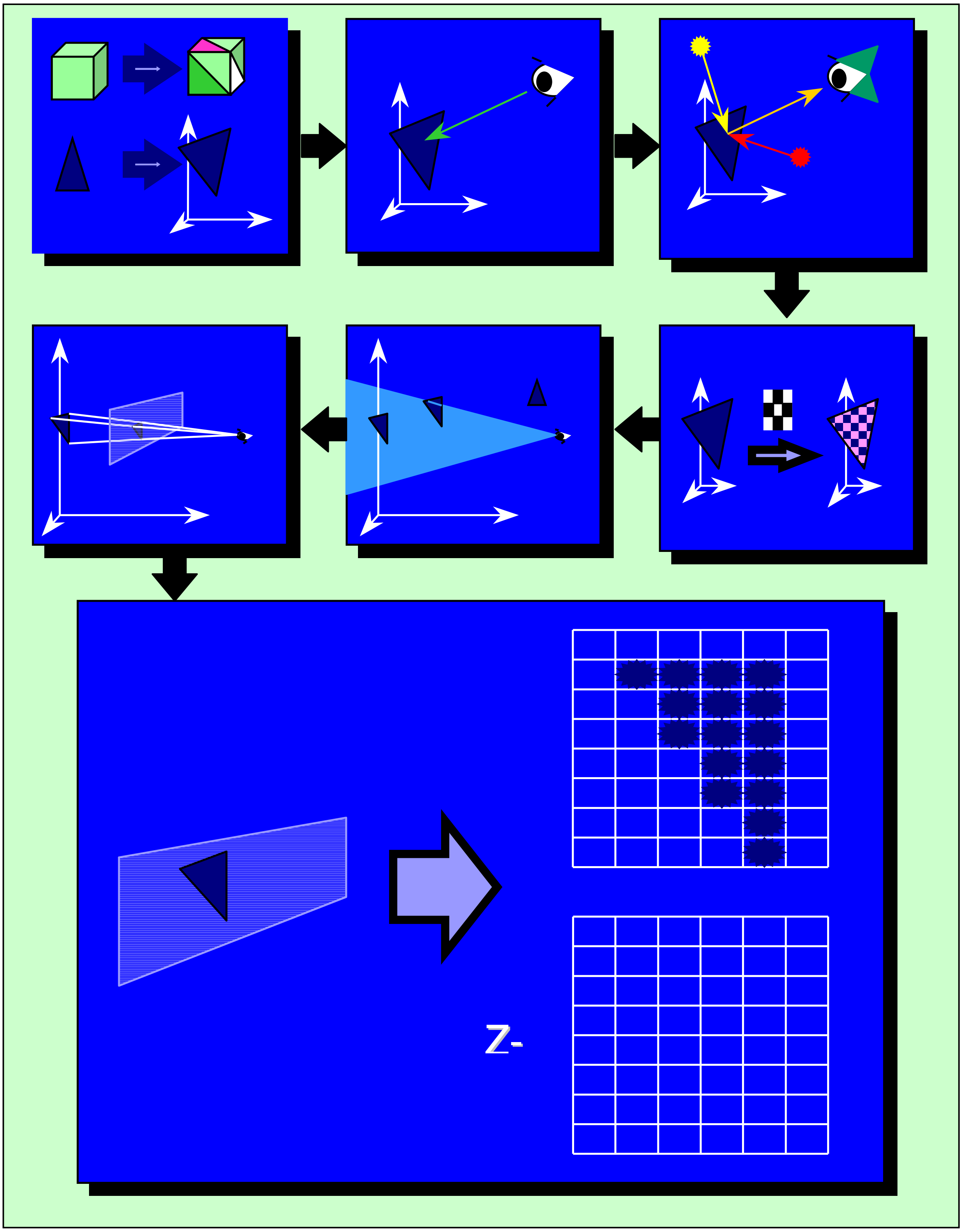
3) Beleuchtung: In dieser Phase wird die Farbe des Dreiecks bestimmt. Dabei werden dessen Eigenfarbe, Leuchtkraft, Durchsichtigkeit, Reflektionseigenschaften, sowie die Farbe der Lichtstrahlen, die auf das Dreieck fallen, beachtet. Die resultierende Farbe wird mit Hilfe von Beleuchtungsgleichungen ermittelt, die wenig mit der physikalischen Wirklichkeit zu tun haben, sich aber in der Praxis bewährt haben, d.h. sie liefern realistische Farbeffekte.

4) Texturen: Zweidimensionale Grafiken werden auf die Oberfläche des Objekts abgebildet. Bei geeigneter Wahl dieser Grafiken erhält der Betrachter den Eindruck einer dreidimensionalen Oberfläche, also einer Oberfläche mit Vertiefungen bzw. Ausbuchtungen. In dieser Phase werden die Eckpunkte auf Koordinaten der Textur abgebildet (texture mapping).

5) Clipping: In dieser Phase werden alle Dreiecke ausgefiltert, die nicht im Blickfeld des Betrachters liegen.

6) Projektion: Das Dreieck wird nun auf die Betrachtungsebene projiziert. Dazu berechnet man die Schnittpunkte der Betrachtungsebene mit den Verbindungsgeraden zwischen den Eckpunkten des Dreiecks und dem Betrachter.

7) Rastern: Bisher war jedes Dreieck nur durch seine Eckpunkte gegeben. Jetzt wird über die Betrachtungsebene ein Raster gelegt, dessen Auflösung meist der des Bildschirms entspricht. Für jeden Rasterpunkt (Pixel = picture element) wird geprüft, ob er innerhalb des Dreiecks liegt. Falls dies zutrifft, wird er in der Farbe des Dreiecks eingefärbt. Dabei wird in einem zweiten Raster, dem Z-Puffer, der Abstand des Punktes vom Betrachter gespeichert. Soll ein Pixel neu eingefärbt werden, wird zuerst abgetestet, ob der entsprechende Punkt des Dreiecks näher beim Betrachter liegt, als der aktuell im Z-Puffer eingetragene Wert. Man testet also punktweise die Überdeckung eines Objekts durch ein anderes.



Virtual Reality Modeling Language – VRML

Durch die immer bessere Hardware ist es heute nicht mehr nötig, spezielle Grafik-Workstations zu verwenden. Auf modernen PCs kann jeder durch dreidimensionale Welten fliegen. Um solche Welten zu definieren und sie über das Internet zu verbinden, wurde die Sprache VRML entwickelt. In diesem Kapitel werden die grundlegenden Sprachkonzepte von VRML zur Beschreibung statischer, dreidimensionaler Objekte betrachtet. Das nächste Kapitel beschäftigt sich dann mit den Mitteln zur Interaktion und Animation solcher Objekte.

Geschichte von VRML

Im Frühling 1994 diskutierte auf der ersten WWW-Konferenz in Genf eine Arbeitsgruppe über "Virtual Reality"-Schnittstellen für das WWW. Es stellte sich heraus, daß man eine standardisierte Sprache zur Beschreibung von 3D-Szenen mit Hyperlinks brauchte. Diese Sprache erhielt in Anlehnung an HTML zuerst den Namen Virtual Reality Markup Language. Später wurde sie in Virtual Reality Modeling Language umbenannt. Die VRML-Gemeinde spricht die Abkürzung gerne "Wörml" aus. Basierend auf der Sprache Open Inventor von Silicon Graphics (SGI) wurde unter der Federführung von Mark Pesce die Version 1.0 von VRML entworfen. Im Laufe des Jahres 1995 entstanden eine Vielzahl von VRML-Browsern (u.a. WebSpace von SGI, Live3D PlugIn von Netscape). Die virtuellen Welten, die man mit VRML 1.0 spezifizieren kann, sind zu statisch. Zwar kann man sich mit einem guten VRML-Browser flott und komfortabel durch diese Welten bewegen, aber die Interaktion ist auf das Anklicken von Hyperlinks beschränkt. Im August '96, anderthalb Jahre nach der Einführung von VRML 1.0, wurde auf der SIGGraph'96 die Version VRML 2.0 vorgestellt. Sie basiert auf der Sprache Moving Worlds von Silicon Graphics. Sie ermöglicht Animationen und sich selbstständig bewegende Objekte. Dazu mußte die Sprache um Konzepte wie Zeit und Events erweitert werden. Außerdem ist es möglich, Programme sowohl in einer neuen Sprache namens VRMLScript oder in den Sprachen JavaScript oder Java einzubinden. Ende 1997 wurde eine überarbeitete Version von VRML 2.0 zum offiziellen ISO-Standard VRML97.

Was ist VRML?

Die Entwickler der Sprache VRML sprechen gerne von virtueller Realität und virtuellen Welten. Diese Begriffe scheinen mir aber zu hoch gegriffen für das, was heute technisch machbar ist: Eine grafische Simulation dreidimensionaler Räume und Objekte mit eingeschränkten Interaktionsmöglichkeiten. Die Idee von VRML besteht darin,solche Räume über das WWW zu verbinden und mehreren Benutzern gleichzeitig zu erlauben, in diesen Räumen zu agieren. VRML soll architekturunabhängig und erweiterbar sein. Außerdem soll es auch mit niedrigen Übertragungsraten funktionieren. VRML ist ein Dateiformat, mit dem man interaktive, dynamische, dreidimensionale Objekte und Szenen speziell fürs World-Wide-Web beschreiben kann. Im WWW erscheinen Daten und Dienste des Internets als ein gigantisches, verwobenes Dokument, in dem der Benutzer blättern kann. Mit VRML sollen die Daten und Dienste des Internets als ein riesiger Raum, ein riesiges Universum erscheinen, in dem sich der Benutzer bewegt -- als der Cyberspace.

Konzepte von VRML 2.0

Eine zentrale Rolle in 3D-Grafiksystemen spielt der Szenengraph. Die Anzeigesoftware traversiert diesen Graphen und akkumuliert entlang eines jeden Pfades, die in den Knoten enthaltenen Informationen, etwa Transformationen, Farbinformation und geometrische Daten. Ein Knoten kann über verschiedene Pfade erreicht werden, d.h. er kann in unterschiedlichen Kontexten ausgewertet werden. In klassischer 3D-Software wurde der Szenengraph im Programmcode des 3D-Systems selbst aufgebaut, er war also in das 3D-System integriert. Mit Hilfe von VRML kann ein Szenengraph unabhängig von der Implementierung des Grafiksystems angegeben werden. Die Konstrukte von VRML heißen in Anlehnung an den durch sie spezifizierten Szenengrahen Knoten und Felder, wobei Felder den Kanten im Graphen entsprechen. Einen Knoten definert man durch Angabe des Knotentyps und der Werte für die Felder des Knotens. Diese Werte können wieder Knoten sein. Es gibt Defaultwerte für die Felder jeden Knotentyps, so daß man nicht immer alle Felder angeben muß. Die Syntax ist wie folgt: `Knotentyp { feld`$_1$ `wert`$_1$ … `feld`$_n$ `wert`$_n$ `}`

Jedes Feld eines Knotens akzeptiert nur Werte eines bestimmten Typs. Bei Feldern unterscheidet man verschiedene Arten: field, exposedField, eventIn und eventOut, darauf wird aber erst etwas später genauer eingegangen. In VRML gibt es einfache und mehrfache Werte. Mehrfache Werte werden in eckigen Klammern als durch Kommata voneinander getrennte Elemente einer Liste angegeben. Enthält die Liste nur einen Wert, kann man auf die eckigen Klammern verzichten. Die Typen von VRML 2.0 sind in der folgenden Tabelle aufgeführt. Einfache Typen haben den Präfix `SF`, mehrfache den Präfix `MF`. Überall wo mehrfache Typen als Wert erlaubt sind, können auch einfache verwendet werden, aber nicht umgekehrt.

Typ	Beschreibung	Beispiel
SFBool	Wahrheitswert	TRUE oder FALSE
SFColor / MFColor	Drei Gleitkommazahlen zwischen 0 und 1, die den Anteil der Farben Rot, Grün und Blau angeben.	1.0 0.5 0.3
SFFloat / MFFloat	Gleikommazahl	3.1416
SFImage	Bild, das als Folge von Graustufen oder Farbwerten angegeben ist. Format: m n d $p_1 \ldots p_{n*m}$ Wobei m die Breite, n die Höhe und d die Anzahl der Bytes pro Farbinformation ist.	3 3 1 0x00 0xFF 0x00 0xFF 0xFF 0xFF 0x00 0xFF 0x00
SFInt32 / MFInt32	Ganze Zahl (32 Bit)	3014 oder 0xFF03FFA0
SFNode / MFNode	Ein VRML-Knoten	Box { } oder USE Kugel
SFRotation / MFRotation	Vier Gleitkommazahlen, die ersten drei geben die Rotationsachse an (vom Ursprung zu dem angegebenen Punkt), die vierte den Winkel, um den gedrehte wird.	0 1 0 3.1416

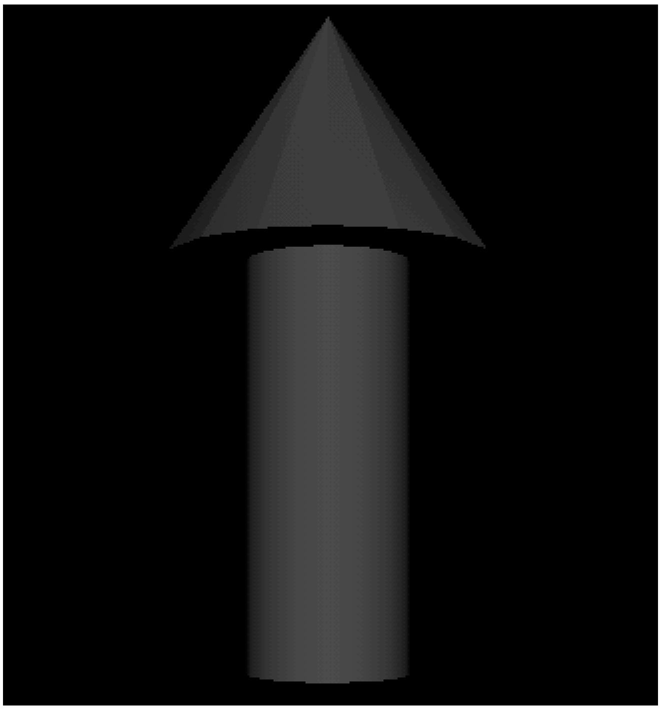
SFString / MFString	Zeichenkette	“Beispiel”
SFTime / MFTime	Zeitpunkt, als Anzahl der Sekunden nach dem 1.1.1970 00:00:00 Uhr GMT	
SFVec2f / MFVec2f	Vektor aus zwei Gleitkommazahlen	3.0 4.0 oder 3.0 , 4.0
SFVec3f / MFVec3f	Vektor aus drei Gleitkommazahlen	3.0 4.0 5.7 oder 3.0 , 4.0 , 5.7

VRML: Ein Beispiel

Was VRML von anderen Objektbeschreibungssprachen unterscheidet,ist die Existenz von Hyperlinks, d.h. durch Anklicken von Objekten kann man in andere Welten gelangen oder Dokumente wie HTML-Seiten in den WWW-Browser laden. Es ist auch möglich, Grafikdateien, etwa für Texturen, oder Sounddateien oder andere VRML-Dateien einzubinden, indem man die URL, d.h. die Adresse der Datei im WWW angibt. Eine VRML-Datei erkennt man an ihrer Kopfzeile “`#VRML V2.0 utf8`”. In ihr sind sowohl die verwendete VRML-Version, als auch die verwendete Zeichenkodierung angegeben. Dies ist derzeit UTF8 und war für Version 1.0 ASCII.

```
#VRML V2.0 utf8

DEF APP Appearance {
         material Material { diffuseColor 1 0 0 } }

Shape { appearance USE APP
        geometry Cylinder { radius 1 height 5 } }

Anchor { children
   Transform { translation 0 4 0
      children Shape { appearance USE APP
                       geometry Cone { bottomRadius 2
                                       height 3 } } }
  url "anotherWorld.wrl" }
```

In diesem Beispiel definieren wir einen Pfeil. Wenn man die Pfeilspitze (Zylinder) anklickt, wird die Datei `anotherWorld.wrl` geladen.



Statische Szenen

Ein dreidimensionales Objekt in VRML wird durch drei Eigenschaften beschrieben: seine Geometrie, sein Erscheinungsbild (Oberfläche) und Transformationen seiner Position, Ausrichtung und Skalierung.

Geometrische Knoten

In VRML verwendet man als Maßeinheiten für Längen Meter und für Winkel das Bogenmaß (Radiant). Geometrische Knoten legen die räumliche Ausdehnung eines Objekts fest. Es gibt geometrische Knoten für Kugeln, Kegel, Zylinder und Quader, sowie für Text:

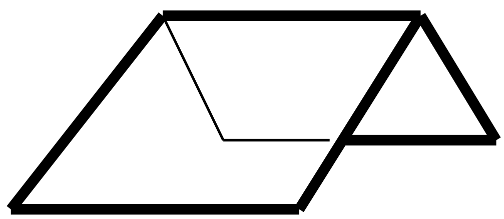
- `Sphere { radius 0.5 }`
  Eine Kugel mit Radius 0.5m wird erzeugt. Der Ursprung des lokalen Koordinatensystems liegt im Mittelpunkt der Kugel.
- `Cone { bottomRadius 3 height 5 }`
  Ein Kegel mit Radius 3m und Höhe 5m wird erzeugt. Der Urprung des lokalen Koordinatensystems liegt in der Mitte der Mittelsenkrechten.
- `Cylinder { radius 3  height 5 }`
  Ein Zylinder mit Radius 3m und Höhe 5m wird erzeugt. Der Urprung des lokalen Koordinatensystems liegt in der Mitte der Mittelsenkrechten.
- `Box { size 0.5 2 3 }`
  Ein Quader mit Breite 0.5m, Tiefe 2m und Höhe 3m wird erzeugt. Der Urprung des lokalen Koordinatensystems liegt in der Mitte des Quaders, nämlich der Schnittpunkt der Diagonalen.
- `Text { string "Beispieltext"`

```
        fontstyle FontStyle { size 2 family "SANS" style "BOLD"
} }
```

Auch eine Zeichenkette kann als dreidimensionales, grafisches Objekt definiert werden. Für größere Textpassagen ist dies allerdings nicht zu empfehlen, da der Berechnungsaufwand für die Anzeigesoftware sehr hoch ist. In diesem Fall bindet man den Text besser als 2D-Grafik, d.h. als Textur, ein.

Außerdem gibt es geometrische Knoten, mt denen man Objekte direkt durch Angabe ihrer Eckpunkte und der sie verbindenden Linien angibt, zu diesen Knoten gehören `IndexedLineSet, IndexedFaceSet,PointSet`. Als Beispiel dient im Folgenden der Knoten `IndexedFaceSet`, bei dem ein Objekt durch Angabe seiner Begrenzungsflächen definiert wird.

```
IndexedFaceSet {
  coord Coordinate {
        point [ -1 0  1,
                -1 0 -1,
                1  0 -1,
                1  0  1,
                1  1  0,
                -1  1  0 ] }
  coordIndex [ 0,1,2,3,-1,
               0,3,4,5,-1,
               1,2,4,5,-1 ] }
```



Im Knoten vom Typ `Coordinate` werden die Eckpunkte des Objekte angegeben, d.h. der in der Abbildung mit 0 markierte Punkt hat z.B. die Koordinaten –1 0 1. Im Feld `coordIndex` werden nun die Begrenzungsflächen durch Aufzählung ihrer Eckpunkte angegeben. Wie schon oben diskutiert wurde, unterscheidet man bei Flächen Vorder- und Rückseiten. In VRML verwendet man die Konvention, daß man die Punkte einer Begrenzungsfläche im Gegenuhrzeigersinn aufzählt bei Blick auf die Vorderseite der Fläche. Zur Referenz auf einen Punkt verwendet man dessen Position in der Liste des `Coordinate`-Knoten beginnend bei 0. Hinter dem letzten Punkt einer Fläche schreibt man eine –1.

Damit der VRML-Browser ein Objekt wirklich anzeigen kann, muß das Objekt durch einen `Shape`-Knoten definiert werden. Ein solcher Knoten legt die Geometrie des Objekts und das Erscheinungsbild seiner Oberfläche fest.

```
Shape {
  geometry Box { size 1 4 2 }
  appearance Appearance { material  ...
                          texture ...
                        }
}
```

Erscheinungsbild

Das Erscheinungsbild der Oberfläche wird in einem Knoten des Typs `Appearance` angegeben. Als Oberfläche eines Objekts kann eine zweidimensionale Grafik, aber auch durch eine animierte Grafik (MPEG-Movie) verwendet werden. Diese sogenannten Texturen gibt man als Wert des Feldes `texture` an. Am einfachsten jedoch ist es, eine Farbe für die Oberfläche im Feld `material` anzugeben. Die Oberflächenfarbe gibt man in einem Knoten des Typs `Material` an. Das Feld `diffuseColor` gibt die Farbe des vom Objekt reflektierten Lichts an. Das Feld `ambientIntensity` gibt an, wieviel Licht das Objekt reflektieren soll. Für Glanzeffekte wird die Farbe im Feld `specularColor` verwendet, die Stärke des Effektes wird im Feld `shininess` angegeben. Schließlich kann regelt der Wert des Feldes `transparancy` , wie durchsichtig die Oberfläche des Objekts sein soll, ein Wert von 1.0 bedeutet ganz durchsichtig, einer Wert von 0.0 bedeutet undurchsichtig.

```
Material
 { diffuseColor  0.2 0.3 0.8
   ambientIntensity 0.2
   emissiveColor   0 0.3 0.1
   shininess 0.8
   transparency 0.8  }
```

Um eine 2D-Grafikdatei als Textur anzugeben, verwendet man einen Knoten des Typs `ImageTexture`. Als Wert des Feldes `url` gibt man die URL der Grafikdatei an. Die Werte der Felder `repeatS` und `repeatT` geben an, ob die Textur gestreckt wird, um das Objekt in horizontaler bzw. vertikaler Richtung zu überdecken oder ob es vervielfacht wird.

```
ImageTexture
 { url "stephan.jpg"
   repeatS TRUE
   repeatT TRUE
  }
```

Außerdem kann man auch eine Textur explizit punktweise definieren in einem Knoten des Typs `PixelTexture`, oder man kann MPEG-Videodateien mit Hilfe eines `MovieTexture`-Knotens als Textur verwenden.

Transformationen

Nachdem man ein Objekt oder ein Teilobjekt definiert hat, muß man es im Raum plazieren. Dazu kann man es von bzgl. des umgebenden Koordinatensystems um den Wert des Feldes `translation` verschieben, um die im Feld `rotation` angegebene Achse um den dort ebenfalls angegebenen Winkel rotieren und man kann es um den im Feld `scale` angegebenen Wert stauchen bzw. strecken. Im Feld `children` können ein oder mehrere Objekte angegeben werden.

```
Transform { children Shape { ... }
            center 0 0 0
            translation 0 1 0
            rotation 0 1 0 1.57
            scale 1 2 1
            scaleOrientation 0 0 1 0
          }
```

Betrachtet man die zugehörigen Transformationsmatrizen C (center), T (translation), R (rotation), S (scale) und SR (scaleOrientation), dann wird der Punkt P folgendermaßen in den Punkt P' abgebildet:

$$P' = P * T * C * R * SR * S * SR^{-1} * C^{-1}$$

Praktisch heißt dies, daß zuerst der Mittelpunkt verschoben, dann die Rotation für die Skalierung durchgeführt, dann skaliert, dann rotiert und schließlich verschoben wird.

Transformationen können auch verschachtelt werden. Dies verwendet man vor allem um Objekte aus mehreren Teilobjekten zusammenzubauen. Manchmal ist es aber auch notwendig Transformationen zu verschachteln, wenn man von der oben angegebenen Reihenfolge der Komposition der Teiltransformationen abweichen will.

```
Transform { translation 0 1 0
            Children Transform { translation 0 0 1
                                   children Shape { ... }
                                 }
          }
```

Weitere (statische) Knoten

Es gibt noch eine Vielzahl anderer Knotentypen für statische VRML-Szenen, von denen hier noch einige kurz beschrieben werden:

- `LOD`: In diesem Knoten kann man mehrere Objekte unterschiedlicher Detailgenauigkeit angeben. In Abhängigkeit der Entfernung des Betrachters von diesem Knoten, kann der VRML-Browser dann ein Objekt mit entsprechender Detailgenauigkeit wählen (level of detail).
- `Switch`: In diesem Knoten kann man mehrere Objekte angeben und über ein Feld eines oder keines dieser Objekte auswählen, das vom Browser dann dargestellt werden soll.
- `Anchor`: Klickt der Betrachter ein Objekt an, das unter diesem Knoten im Szenengraph hängt, dann wird die in diesem Knoten angegebene Datei geladen.
- Lichtquellen, z.B. `SpotLight`: Man kann Lichtquellen im Raum plazieren und die Richtung, Farbe und Intensität ihrer Lichtstrahlen bestimmen.
- `Inline`: Mit diesem Knoten wird der in einer anderen VRML-Datei angegebene Szenengraph an dieser Stelle im Szenengraph eingebunden.
- `Background`: Für verschiedene Teile des Hintergrunds kann man in diesem Knoten Farben oder Grafikdateien angeben.
- `Viewpoint`: Mit diesem Knoten definiert man einen Punkt, von dem aus der Betrachter auf die Szene schauen kann, also verschiedene Kamerapositionen, von denen der Betrachter eine wählen kann.

DEF/USE

Mit den bisher vorgestellten Mitteln könnte man in VRML nur Szenenbäume definieren, d.h. jeder Knoten hätte nur eine eingehende Kante. Mit DEF und USE kann man aber einen Knoten wiederverwenden, d.h. an mehreren Stellen im Szenengraph einbauen. Mit `DEF` *`DefName KnotenTyp`* `{ }` bindet man den durch *`Knotentyp`* { } spezifizierten Teilszenengraph an den Namen *`DefName.`* An anderer Stelle in der Spezifikation kann nun dieser Szenengraph wieder mit `USE` *`DefName`* referenziert werden.

```
DEF Kugel Shape { geometry Sphere { radius 3 } }
Transform { translation 0 2 0
            children USE Kugel
          }
```

Erweiterbarkeit

Prototypen erlauben es neue Knotentypen zu definieren. Ein solcher Knotentyp kann Felder, einschließlich neuer Ein- und Ausgabeevents definieren. In diesem Beispiel wird ein neuer Knotentyp `Kugel` definiert. Er ist mit der Farbe der Kugel parametrisiert: in eckigen Klammern werden die nach außen sichtbaren Felder des Knotens angegeben, hier `col`. Zuerst wird die Feldart, hier `field` (es gibt auch `eventIn`, `eventOut` und `exposedField`, aber mehr dazu später), dann der Typ des Feldes, hier `SFColor`, dann der Feldname, hier `col`, und schließlich ein Defaultwert, hier `0 0 0.2`, für das Feld angegeben.

```
PROTO Kugel [ field SFColor col 0 0 0.2 ]
{ Shape { geometry Sphere { radius 3 }
          appearance Appearance {
                        material Material { diffuseColor IS col }
                       }
        }
}

Kugel { col 0 1 0.5 }
```

Angenommen, dieser Protoyp wurde in der Datei protos.wrl definiert, dann kann man ihn mit `EXTERNPROTO` importieren:

```
EXTERNPROTO Kugel []
            http://www.somewhere.else/~someone/protos.wrl
```

In der VRML Spezifikation sind Prototypen für alle Standardknoten, also etwa `Transfrom`, `Shape`, `Sphere`, usw., angegeben – allerdings mit leerem Rumpg `{ }`, da die Implementierung dieser Knoten ja direkt durch den VRML-Browser erfolgt.

Dynamisches VRML

VRML bietet verschiedene Techniken, mit denen man interaktive Animationen erstellen kann. Diese Techniken unterscheiden u.a. dadurch, ob sie zusätzliche Programmierung erfordern. Im Folgenden werden drei Techniken vorgestellt. Zuerst das Erstellen von Animationen mit VRML allein, d.h. mit Hilfe von Events, Routes, Sensoren und Interpolatoren. Dann wird die Verwendung von Script-Knoten betrachtet, die Programmteile in JavaScript/VRMLScript oder Java einkapseln und zum Abschluß die Steuerung einer VRML-Szene durch ein Applet über das External Authoring Interface (EAI).

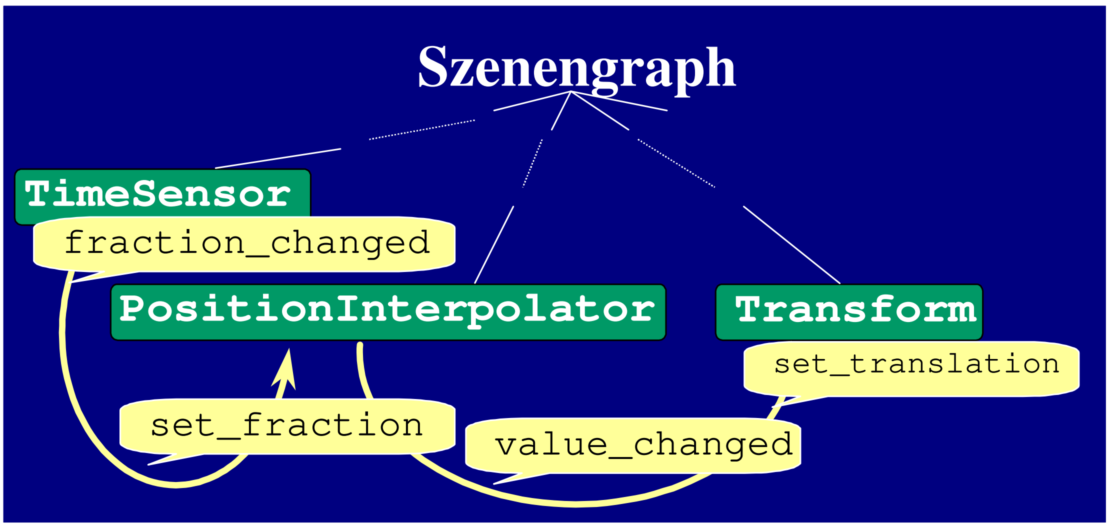
Events, Routes, Sensoren und Interpolatoren

Außer auf Anklicken von Hyperlinks können VRML-Welten auf eine Reihe weiterer Ereignisse reagieren. Dazu wurden sogenannte Sensoren eingeführt. Sensoren erzeugen Ausgabe-Events aufgrund externer Ereignisse wie Benutzeraktionen oder nach Ablauf eines Zeitintervalls. Events können an andere Objekte geschickt werden, dazu werden die Ausgabe-Events von Objekten mit den Eingabe-Events anderer Objekte durch sogenannte Routes verbunden. Ein Sphere-Sensor zum Beispiel wandelt Bewegungen der Maus in 3D-Rotationswerte um. Ein 3D-Rotationswert besteht aus drei Zahlenwerten, die die Rotationswinkel in Richtung der drei Koordinatenachsen angeben. Ein solcher 3D-Rotationswert kann an ein anderes Objekt geschickt werden, das daraufhin seine Ausrichtung im Raum entsprechend verändert. Ein anderes Beispiel für einen Sensor ist der Zeitsensor. Er kann z.B. periodisch einen Event an einen Interpolator schicken. Ein Interpolator definiert eine abschnittsweise lineare Funktion, d.h. die Funktion ist durch Stützstellen gegeben und die dazwischenliegenden Funktionswerte werden linear interpoliert. Der Interpolator erhält also einen Eingabe-Event e vom Zeitsensor, berechnet den Funktionswert f(e) und schickt nun f(e) an einen anderen Knoten weiter. So kann ein Interpolator zum Beispiel die Position eines Objekts im Raum in Abhängigkeit von der Zeit festlegen. Dies ist der grundlegende Mechanismus für Animationen in VRML.

Bisher wurde nur gesagt, daß Knoten über Felder verfügen. Es gibt in VRML allerdings verschiedene Arten von Feldern: Felder i.e.S (field), Eingabe-Events (eventIn) und Ausgabe-Events (eventOut). Außerdem gibt es eine Kombination aller drei, ein sogenanntes exposedField. Fields erhalten ihren Wert beim Instantiieren des Knotens und dieser Wert kann anschließend nicht geändert werden. EventIns erhalten ihren Wert über Routes von anderen Knoten. EventOuts senden ihren Wert über Routes an andere Knoten. ExposedFields werden beim Instantiieren wie Fields verwendet, können aber auch ihren Wert über Routes von anderen Knoten erhalten oder ihren Wert an andere schicken. Für ein ExposedField mit Namen `translation` zum Beispiel wird automatisch ein EventIn `set_translation` und ein EventOut `translation_changed` angelegt. Routes können überall in einer VRML-Datei angegeben werden. Man verwendet dazu folgende Syntax:

`ROUTE` *`DefName1.EventOutName`* `TO` *`DefName2.EventInName`*

Dabei bezeichnen *`DefName1`* und *`DefName2`* Knoten, die zuvor mit `DEF` an diese Namen gebunden wurden und *`EventOutName`* und *`EventInName`* sind Namen von Ein- bzw. Ausgabe-Events dieser Knoten.



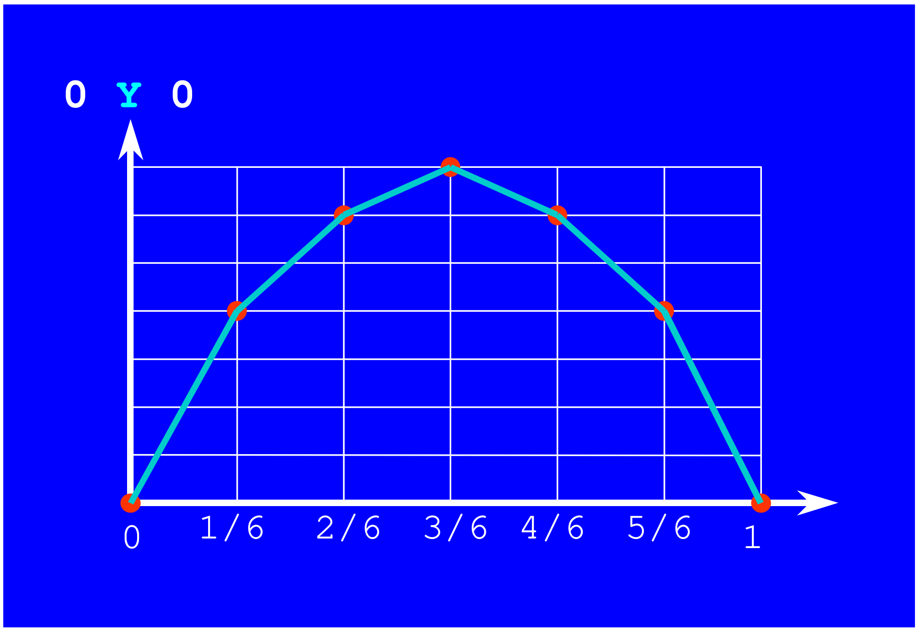
Beispiel: Hüpfender Ball

In diesem Beispiel wird eine Kugel mit dem oben beschriebenen Prototyp erzeugt und mit einem `Transform`-Knoten plaziert, d.h. hier durch die Defaultwerte des `Transform`-Knotens im Ursprung des Koordinatensystems. Dann wird eine Uhr definiert, sie läuft vom Laden der Datei an in einer `Schleife (loop TRUE)` und erzeugt periodisch `fraction_changed` Ausgabe-Events vom Typ `SFFloat`. Der Wert dieses Events durchläuft zyklisch das Intervall [0..1]. In dem Beispiel wird dieser Event and einen `PositionInterpolator` als Eingabe-Event `set_fraction` geschickt. Der `PositionInterpolator` erzeugt den Ausgabe-Event `value_changed` vom Typ `SFVec3f`, d.h. 3D Koordinaten. Dieser Event wird nun an den `Transform`-Knoten, der an BALL gebunden ist, als `set_translation` Eingabe-Event geschickt.

```
DEF BALL Transform { children Kugel { } }

DEF CLOCK TimeSensor { loop TRUE cycleInterval 3.5 }

DEF POSITION
 PositionInterpolator
  {  key      [ 0,     0.166, 0.333, 0.499, 0.666, 0.833, 1 ]
     keyValue [ 0 0 0, 0 4 0, 0 6 0, 0 7 0, 0 6 0, 0 4 0, 0 0 0  ]  }

ROUTE CLOCK.fraction_changed TO POSITION.set_fraction
ROUTE POSITION.value_changed TO BALL.set_translation
```

PositionInterpolator

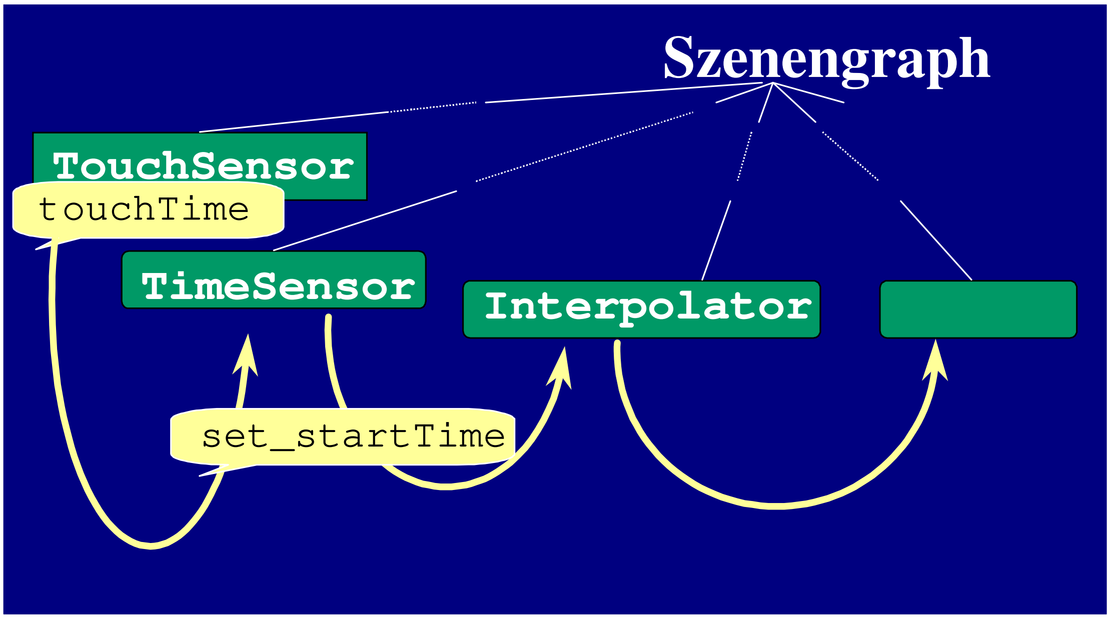
In der folgenden Abbildung ist die in unserem Beispiel interpolierte Funktion dargestellt. Die Funktion ist auf dem Intervall [0..1] definiert. Es wird linear interpoliert. Die Stützstellen werden als Wert des Feldes `key` angegeben und die Funktionswerte an den Stützstellen als Wert des Feldes `keyValue`.



Animationen kann man auch erst in Abhängigkeit von Benutzerinteraktionen starten. Das folgende Beispiel erweitert obige Animation, so daß sie erst gestartet wird, wenn der Ball angeklickt wurde. Dazu wird ein TouchSensor-Knoten verwendet. Der TouchSensor-Knoten wird als zusätzliches Kind als Wert des Feldes children eines Transform-Knotens eingetragen. Wenn eines der anderen Kinder dieses Transform-Knotens angeklickt wird, dann löst der eingetragene TouchSensor einen Ausgabe-Event touchTime vom Typ SFTime aus. Im Beispiel wird dieser als Eingabe-Event set_startTime an den TimeSensor geschickt, der erst daraufhin beginnt, Events zu erzeugen.

```
DEF BALL
 Transform { children [ DEF SCHALTER TouchSensor { }
                        Kugel { }
                      ] }

DEF CLOCK TimeSensor { loop FALSE cycleInterval 3.5 }

DEF POSITION
 PositionInterpolator
  {  key [ 0, 0.166, 0.333, 0.499, 0.666, 0.833, 1 ]
     keyValue [ 0 0 0,   0 4 0,   0 6 0,   0 7 0,
                0 6 0,   0 4 0,   0 0 0  ]  }

ROUTE CLOCK.fraction_changed TO POSITION.set_fraction
ROUTE POSITION.value_changed TO BALL.set_translation
ROUTE SCHALTER.touchTime TO CLOCK.set_startTime
```

Nach dem gleichen Prinzip kann man auch Animationssequenzen implementieren. Jede Teilanimation wird durch eine eigene Uhr getaktet, am Ende eines Uhrlaufs wird die Uhr für die nächste Teilanimation gestartet.



Andere Sensoren und Interpolatoren

- CylinderSensor: Bewegungen der Maus werden in Rotationen um die lokale Y-Achse übersetzt.

- PlaneSensor: Bewegungen der Maus werden in Verschiebungen in lokale X- und Y-Richtung übersetzt.

- ProximitySensor: Betritt, verläßt oder bewegt isch der Benutzer im Bereich dieses Sensors, dann schickt der Sensor Events an andere Knoten.

- SphereSensor: Bewegungen der Maus werden in Rotationen um den lokalen Mittelpunkt übersetzt.

- VisibilitySensor: Kommt das Objekt ins Blickfeld des Betrachters oder verläßt er dieses, dann erzeugt dieser Sensor Events.

- ColorInterpolator (SFColor): Dieser Knoten definiert eine abschnittsweise linear Funktion auf Farbwerten.

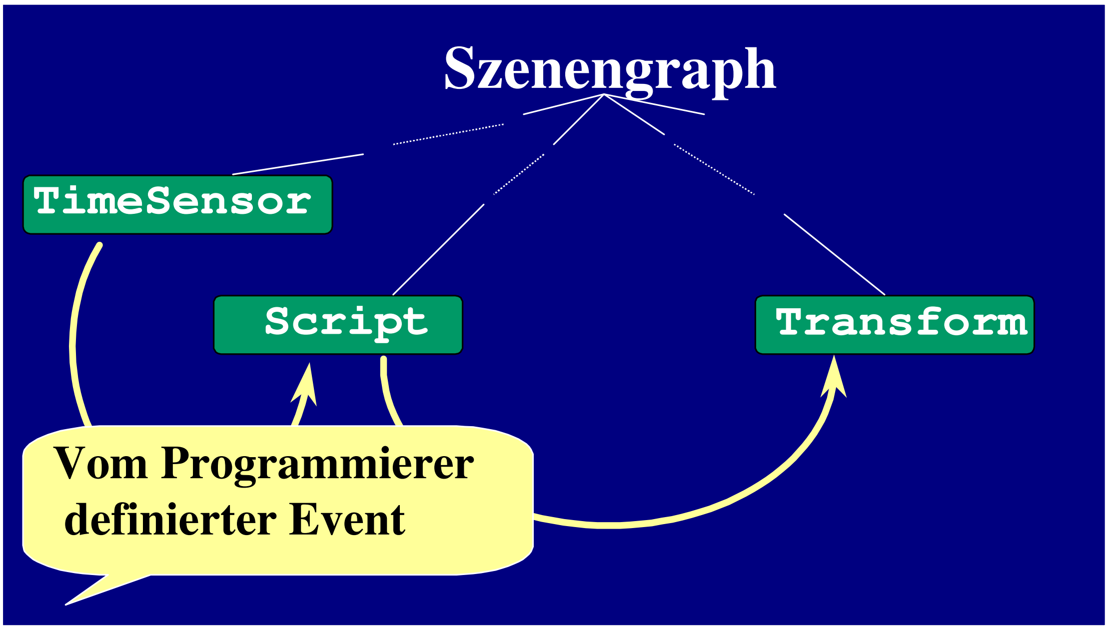
- CoordinateInterpolator (MFVec3f): Dieser Knoten definiert eine abschnittsweise lineare Funktion auf auf Tripeln von Gleitkommazahlen (Koordinaten). Im Gegensatz zum PositionInterpolator kann dieser Knoten auch Listen von Koordinaten als Funktionswert liefern.

- OrientationInterpolator (SFRotation): Dieser Knoten definiert eine abschnittsweise lineare Funktion auf Rotationen. Rotationen bestehen aus Koordinaten, die eine Rotationsachse (vom Ursprung zu diesen Koordinaten) festlegen und einem Rotationswinkel.

- ScalarInterpolator (SFFloat): Dieser Knoten definiert eine abschnittsweise lineare Funktion auf Gleitkommazahlen.

Script

Wem Interpolatoren nicht genügend Möglichkeiten bieten, der kann innerhalb von Script-Knoten Funktionen, die in JavaScript oder Java programmiert wurden, einbauen. Ein Script-Knoten unterscheidet sich prinzipiell von anderen Knoten dadurch, daß er neue Felder und Events definieren kann. Die Events dienen der Kommunikation des eingekapselten Programms mit den anderen Knoten im Szenengraph, die Felder dienen dazu, Werte zwischen Aufrufen der im Knoten eingekapselten Funktionen zu speichern. In diesem Sinne ähneln sie globalen Variablen in Programmiersprachen.



Die Syntax eines Script-Knotens ist wie folgt:

```
Script
 { <Feld und Eventdeklarationen>
   directOutput TRUE
   mustEvaluate FALSE
   url <Programm>
 }
```

Die Felder directOutput und mustEvaluate teilen dem Browser mit, ob er bestimmte Optimierungen machen darf. Das Feld directOutput soll den Wert TRUE haben, wenn das eingekapselte Programm außer über die Events des Script-Knotens auch auf andere, für den VRML-Browser nicht sichtbare Weise den Szenengraph verändert. Das Feld mustEvaluate soll den Wert FALSE haben, wenn der Browser die Auswertung der Eingabe-Events an diesen Knoten verzögern kann, bis die Werte der Ausgabe-Events dieses Knotens gebraucht werden.

Beispiel: Kreisender Ball

Wir beschreiben hier nur, wie man Script-Knoten mit JavaScript verwendet. Die Verwendung von Java unterscheidet sich nur unwesentlich hiervon, allerdings muß man anstatt des JavaScript-Programmkodes einen Verweis (URL) auf die Klassendatei mit dem übersetzten Byte-Code für das Java-Programm, eine Unterklasse der Klasse Script, angeben. Die Einsatz von Java wird im nächste Abschnitt über die dritte Technik für dynamische Szenen, nämlich der Verwendung des External Authoring Interface, näher gezeigt.

In einem Script-Knoten schreibt man für jeden Eingabe-Event eine Funktion gleichen Namens. Diese wird aufgerufen, wenn ein neuer Wert für diesen Event über ein Route von einem anderen Knoten ankommt. Die Funktion erhält diesen Wert als Parameter. Innerhalb der Funktion können Ausgabe-Events wie Variablen Werte zugewiesen werden.

In unserem Beispiel haben wir einen Ausgabe-Event newPosition vom Typ SFVec3f. In JavaScript kann dieser Event wie ein dreielementige Array-Variable verwendet werden.

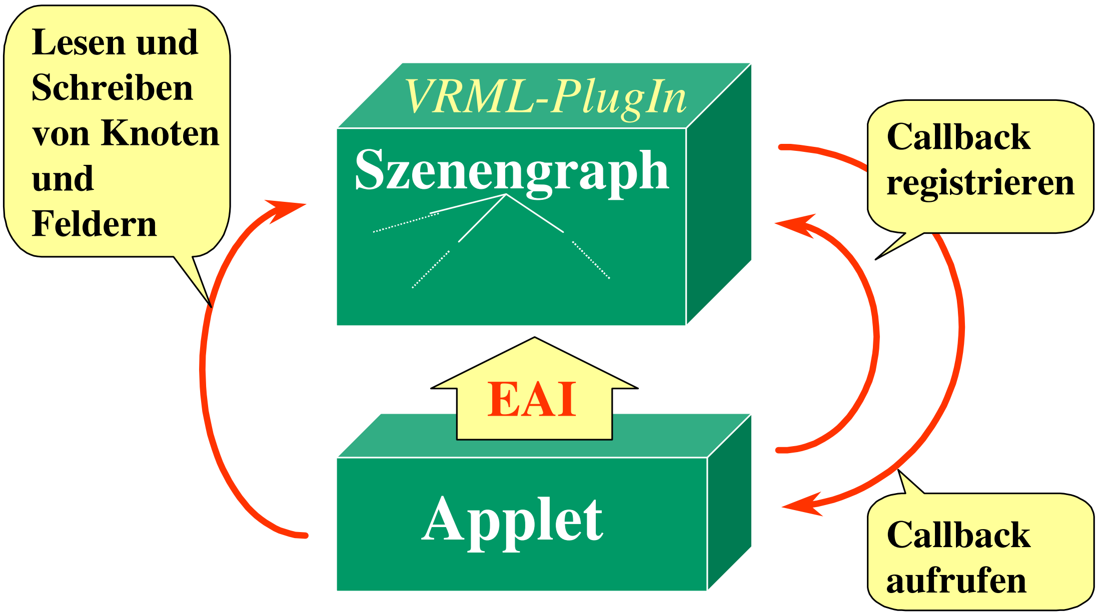
```
DEF ANIMATOR
 Script {
  eventIn SFFloat tick
  eventOut SFVec3f newPosition
  field SFFloat rho 0.0
  url
   "javascript: function tick(val,ts)
    { newPosition[0]=Math.cos(rho)+2*Math.sin(rho/2);
      newPosition[1]=Math.sin(rho);
      newPosition[2]=Math.cos(rho*2)
                     +Math.sin(rho)*Math.cos(rho);
      rho = rho+0.3; }" }

ROUTE CLOCK.fraction_changed TO ANIMATOR.tick
ROUTE ANIMATOR.newPosition TO BALL.set_translation
```

Animationen per External Authoring Interface

Das External Authoring Interface (EAI) ist eine Programmierschnittstelle über die externe Programme mit dem VRML-Browser kommunizieren können. Im folgenden wird die Funktionalität des EAI anhand externer Programme in Form von Java-Applets verdeutlicht. Das EAI erfüllt zwei wichtige Aufgaben:

- Es läßt das externe Programm lesend und schreibend auf den Szenengraph zugreifen.
- Es erlaubt dem externen Programm eigene Funktionen, sogenannte Callbacks, für bestimmte Events anzumelden. Der VRML-Browser ruft diese auf, sobald der entsprechende Event auftritt.

Der Callback-Mechanismus wird manchmal auch scherzhaft als Hollywood-Prinzip bezeichnet. In Hollywood sagen Agenten jungen Schauspielern gerne, sie sollen ihre Telefonnummer hinterlassen für einen späteren Rückruf: “Don’t call us, we’ll call you.”



Beispiel: Ändern des Szenengraphen

Das erste Beispiel besteht aus drei Dateien: einer HTML-Datei, einer VRML-Datei und einer Java-Datei (Applet). Sie zusammen implementieren eine Webseite, in der man die VRML-Beschreibung einer graphischen Szene eintippt, die dann in einem VRML-Browser innerhalb der Seite angezeigt wird. Das Beispiel verwendet HTML, JavaScript, Netscape's LiveConnect, Java, VRML, das EAI und Java. Viele Technik, von der wir hier nur einen geringen Teil erklären werden, wer mehr wissen will, findet in "Java&Co" zu den ersten vier Punkte und wie man sie kombiniert, die wichtigsten Informationen.

Die HTML-Datei

Die VRML-Datei wird mit Hilfe der Markierung `<EMBED>` in die Seite eingebaut, das VRML-PlugIn wird in einem 250*600 Punkte großen Bereich im oberen Teil der Seite angezeigt. Das Applet wird mit `<APPLET>` eingebunden und erhält den Namen `APP`. Das Attribut `MAYSCRIPT` bedeutet, daß das Applet auf Variablen und Methoden von JavaScript und umgekehrt dieses auf die Variablen und Methoden des Applets zugreifen kann. Im Netscape-Browser wird die zugehörige Schnittstelle LiveConnect genannt. Am Ende der Seite wird ein Formular definiert, das den Namen `Formular` erhält. In diesem Formular wird ein Texteingabefeld namens `VRMLSrc` und ein Knopf mit der Aufschrift "Create" angelegt. Wird dieser Knopf angeklickt, so wird die am Anfang der Datei in JavaScript definierte Funktion `createSrc()` aufgerufen. Diese ruft die Methode `addNode()` des Applets auf. Als Parameter erhält die Methode, die im Textfeld `VRMLSrc` eingegebene Zeichenkette.

```
<HTML><HEAD><TITLE> Test EAI </TITLE></HEAD> <BODY>
<SCRIPT>
 function createSrc()
  { var vstr=document.Formular.VRMLSrc.value.toString();
    document.APP.addNode(vstr);
```

```
    return; }
</SCRIPT>
<EMBED SRC="root.wrl" BORDER=0 HEIGHT=250 WIDTH=600>
<APPLET NAME="APP" CODE="EAIApplet.class" MAYSCRIPT
        WIDTH=10 HEIGHT=10> </APPLET>
<FORM NAME="Formular">
Enter a VRML 2.0 Definition of a Children Node:<BR>
<TEXTAREA NAME="VRMLSrc" ROWS=10 COLS=80> </TEXTAREA><BR>
<INPUT TYPE=BUTTON VALUE="Create" ONCLICK="createSrc();" >
</FORM>
</BODY>
</HTML>
```

Die VRML-Datei

In der VRML-Datei wird ein leerer `Transform`-Knoten an den Namen `ROOT` gebunden. Über diesen Namen kann später das Applet auf die Wurzel des Szenengraphen im VRML-Browser zugreifen.

```
#VRML V2.0 utf8

DEF ROOT Transform { }
```

Das Java-Applet

Im Paket vrml.external sind die Klassen und Unterpakete des EAI enthalten. Diese müssen zusätzlich zum Paket java.applet importiert werden. In ter Init()-Methode Browser.getBrowser(), die als Argument das aktuelle Applet erhält, fordert das Applet zuerst ein Objekt (Handle) an, das Zugriff auf das VRML-PlugIn ermöglicht. Dann ruft es die Methode getNode() dieses Objekts auf, um den an den Namen ROOT gebundenen Knoten des im VRML-PlugIn aktuell geladenen Szenengraphen zu erhalten.

```
import java.applet.*;
import vrml.external.field.*;
import vrml.external.Node;
import vrml.external.Browser;

public class EAIApplet extends Applet
{ Browser browser;
  Node root;

  public void init()
   { browser = Browser.getBrowser(this);
     root = browser.getNode("ROOT"); }

  public void addNode(String str)
   { EventInMFNode addChildren =
            (EventInMFNode) root.getEventIn("addChildren");
     Node nodes[] = browser.createVrmlFromString(str);
     addChildren.setValue(nodes); }
 }
```

Beispiel: Callbacks

Wie das vorangegangene Beispiel besteht das folgende auch aus drei Dateien. Diesmal geht es um eine Webseite, auf der in man in einem VRML-Browser eine Kugel verschieben kann. Gleichzeitig wird daneben in einem Applet eine Linie gemäß der Positionsveränderungen der Kugel gezeichnet.

Die HTML-Datei

In der HTML-Datei werden die VRML-Datei und das Applet wie zuvor eingebunden eingebunden.

```
…
<EMBED SRC="Cursor.wrl" BORDER=0 HEIGHT="500" WIDT="400">
<APPLET NAME="APP" CODE="CallbackApplet.class" MAYSCRIPT
        HEIGHT=500 WIDTH=400>
</APPLET>
…
```

Die VRML-Datei

In der VRML-Datei wird in einem Transform-Knoten eine Kugel definiert und eine PlaneSensor. Solange sich der Mauscursor über der Kugel befindet und der Benutzer die Maus bei gedrückter Maustaste bewegt, erzeugt der PlaneSensor translation_changed Ausgabe-Events vom Typ SFVec3f. Diese werden per Route direkt als set_translation Eingabe-Event an den Transform-Knoten weitergeleitet. Dadurch erreicht man, daß sich die Kugel mit dem Mauscursor mitbewegt.

```
#VRML V2.0 utf8

DEF ROOT Transform {
 children [ Shape { geometry Sphere { }
                    appearance Appearance {
                         material Material { diffuseColor 1 0 0 } } }
            DEF PS PlaneSensor { }
         ] }

ROUTE PS.translation_changed TO ROOT.set_translation
```

Das Java-Applet

Das Java-Applet trägt sich selbst im VRML-Browser ein, so daß seine Methode callback() aufgerufen, wenn immer der Wert des Feldes translation des an den Namen ROOT gebundenen Transform-Knoten sich ändert. Zuerst werden die entsprechenden Klassen von Java und des EAI importiert. Anschließend wird die Klasse CallbackApplet definiert. Sie soll die Schnittstelle EventOutObserver des EAI implementieren, d.h. sie muß eine Methode mit der Signatur `public void callback(EventOut, double, Object)` bereitstellen. Wie zuvor wird in der Methode init() je ein Handle auf den VRML-Browser und die Wurzel des Szenengraphen angefordert. Dann liefert die Methode getEventOut() ein Handle auf den Ausgabe-Event translation des Wurzelknotens. Mit der Methode advise() dieses Handles wird eine Callback-Funktion für diesen Event eingetragen, d.h. eigentlich ein Objekt, das die Schnittstelle EventOutObserver implementiert. In diesem Beispiel ist dieses Objekt das Applet selbst (this). Als zweites Argument wird ein Objekt übergeben, das später beim Aufruf der Callback-Methode an diese übergeben wird.

```
import java.applet.*;              // Java APIs
import java.awt.*;
import vrml.external.*;            // VRML APIs
import vrml.external.field.*;
import vrml.external.exception.*;

public class CallbackApplet extends Applet
                            implements EventOutObserver
{ public Browser browser;  // Browser we're using
  Node root;               // Root of the scene graph
  int oldx=150, oldy=150;

  public void init()
  { browser = Browser.getBrowser(this);
    root = browser.getNode("ROOT");
    EventOutSFVec3f translation =
       (EventOutSFVec3f) root.getEventOut("translation");
    translation.advise(this,translation);
  }

  public void callback(EventOut ev, double d, java.lang.Object obj)
  { if (ev.getType()==FieldTypes.SFVec3f)
     {  float[] pos = ((EventOutSFVec3f) ev).getValue();
        int x= Math.round(150+pos[0]*20);
        int y= Math.round(150-pos[1]*20);
        Graphics g = getGraphics();
        g.drawLine(oldx,oldy,x,y);
        oldx=x; oldy=y;
      }
     return;
   }

}
```

Dies kann man dazu verwenden, eine einzelne Callback-Methode bei unterschiedlichen Events einzutragen. Den Aufrufer der Callback-Methode kann man dann über dieses Argument identifizieren. In unserem Beispiel könnte man dies z.B. folgendermaßen in der Callback-Methode realisieren: if ((EventOutSFVec3f) obj)==translation) …

Kritik am EAI

Das EAI ist als Kompromislösung entstanden und hat daher eine stark eingeschränkte Funktionalität. Es sollte nicht mehr Zugriffsmöglichkeiten auf den Szenengraphen erlauben, als bisher über Scripts und Events möglich waren. Dies wurde als notwendig erachtet, damit die Browserhersteller nicht auf Optimierungen des Szenengraphen in ihren Browsern verzichten mußten und es keiner großen Änderungen bedurfte, existierende Browser mit einem EAI zu versehen. Ein wesentlicher Mangel des EAI besteht darin, daß ein Knoten keine Methode kennt, die den Knotentyp des Knotens liefert oder gar alle seine Felder auflistet.

Protokolle für verteile, virtuelle Welten

Damit Rechner überhaupt miteinander kommunizieren können, braucht man Protokolle. In verteilten Systemen kommunizieren Anwendungen durch Austausch von Nachrichten. Ein Protokoll legt Form, Inhalt und Abfolge soclher Nachrichten fest.

Internet-Protokolle

Im folgenden werden zuerst einige grundlegende Protokolle des Internets betrachtet, anschließend werden dann drei Protokolle für verteilte, virtuelle Welten vorgestellt: DIS, DWTP und VRTP.

TCP/IP

Die Abkürzung TCP/IP (Transmission Control Protocol/Internet Protocol) steht eigentlich für eine Sammlung von Protokollen (protocol suite). Diese ermöglicht die Übertragung von Daten über unterschiedlichste Medien wie Koaxialkabel (elektrisch), Glasfaser (optisch) und Richtfunk (elektromagnetisch). Dabei wird nicht ein Protokoll alleine angewandt, sondern die Protokolle werden zu verschiedenen Zwecken und in verschiedenen Ebenen der Kommunikation eingesetzt. Als Beispiel werden im Folgenden die Ebenen des OSI-Referenzmodells (Open Systems Interconnection) näher erläutert. Eine typische Nachricht, die über das Netz geschickt wird, sieht folgendermaßen aus:

DS	NS	TS	SS	RS	AS	Inhalt	DS

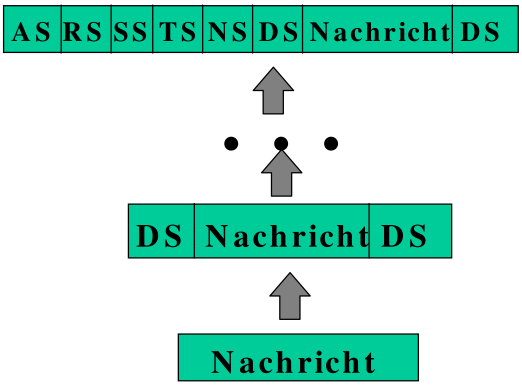
Dabei ist eine Nachricht eine Folge von Nullen und Einsen, die in Segmente

DS

etc. aufgeteilt werden können. Diese Segmente beinhalten Informationen einer bestimmten Protokollschicht. Diese Protokollschichten sind in der folgenden Tabelle aufgeführt.

Schicht	
Anwendungsschicht	beinhaltet Protokolle von Diensten wie ftp, telnet und WWW.
Repräsentationsschicht	kombiniert Daten mit ihren Typen in sog. Records, wichtig für heterogene Netzwerke.
Sitzungsschicht	wird selten unterstützt, regelt, wer gerade senden darf und wer empfangen muß.
Transportschicht	teilt Nachricht in Pakete auf und verteilt für diese Sequenznummern (z.B. TCP)
Netzwerkschicht	fügt Rechneradresse hinzu und sorgt dafür, daß die Nachricht beim richtigen Rechner landet, das sog. Routing (z.B. IP).
Datenschicht	Gruppiert Bits in Rahmen und fügt Prüfsummen hinzu.
Phyiskalische Schicht	überträgt physikalische Repräsentationen von Null und Eins.

Ein Protokoll einer Schicht packt also die Nachricht der darüberliegenden Schicht ein, so wie man einen Brief in einen Umschlag steckt. Auf jeder Schicht kann man das Protokoll wechseln, ohne die darüberliegenden Schichten zu zerstören. Man nimmt einfach die Nachricht aus dem Umschlag (dem vorherigen Protokoll) und steckt sie in einen neuen (das neue Protokoll). Dies ist z.B. wichtig, wenn der Empfänger der Nachricht in einem anderen Netz mit anderen Protokollen liegt. Zwei dieser Protokolle werden jetzt näher erläutert.



Netzwerkschicht: IP (Internet Protocol)

Das IP verwendet 32-Bit-Adressen zur Bezeichnung von Rechnern im Netz. Diese 32 Bits werden in 4 Gruppen zu je 8 Bit unterteilt, den sog. Oktetten. Anstatt die Adresse bitweise anzugeben, schreibt man für jedes Oktett eine Dezimalzahl:

11000001	01100001	11111001	00000100
193	97	251	4

Das Internet ist hierarchisch in Bereiche (engl. domains) aufgeteilt und jedes Oktett entspricht einem solchen Teilbereich. Um die Lesbarkeit solcher Adressen zu erhöhen, schreibt man anstatt der Dezimalzahl auch symbolische Namen. Für das obige Beispiel schreibt man verständlicher: www.addison-wesley.de

Allgemein haben solche Adressen dann die Form:

Computer.Teilnetz.Netz   oder   Computer.Teilnetz.Teilnetz.Netz

Auf der obersten Ebene gibt es z.B. die Netze de für Deutschland, fr für Frankreich, edu für Bildungseinrichtungen in den USA, mil für das amerikanische Militär, org für nichtkommerzielle Organisationen und com für Firmen. Damit man symbolische Bereichsnamen verwenden kann, gibt es spezielle Server, sogenannte Domain Name Server, die Bereichsnamen in IP-Adressen umwandeln.

Theoretisch bieten 32 Bits genug Adressierungsmöglichkeiten für 4 Milliarden Rechner, aber leider wurden einzelne Adreßbereiche pauschal an verschiedene Organisationen wie z.B. das Militär vergeben. Bei dem derzeitigen Wachstum des Internets ist schon abzusehen, daß die verbliebenen Adressen knapp werden. Daher hat man ein abwärtskompatibles Protokoll IPv6 entwickelt, das 128 Bits für Adressen verwendet.

Transportschicht: TCP (Transmission Control Protocol)

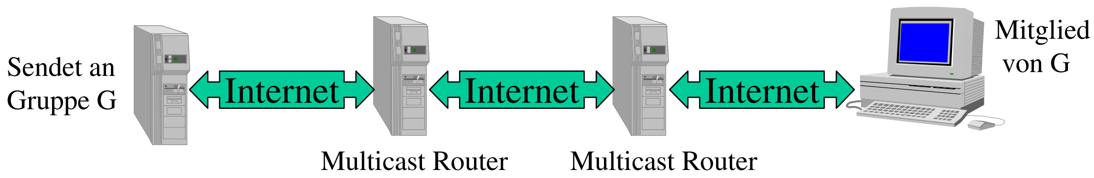
Das IP garantiert weder, daß Pakete in der richtigen Reihenfolge ankommen, noch, daß sie überhaupt ankommen. Dies wird erst durch das TCP ermöglicht. Es zerlegt die Nachricht in Pakete, vergibt Sequenznummern, veranlaßt die Übertragung verlorengegangener Pakete und fügt die Pakete beim Empfänger wieder zu der ursprünglichen Nachricht zusammen. Außer an einen Rechner direkt kann mit TCP auch eine Nachricht an einen Port des Rechners geschickt werden. Diese Ports entsprechen einzelnen Diensten, die der Rechner anbietet. Jeder Port hat eine 16-Bit-Nummer. Üblicherweise ist den Internet-Diensten telnet der Port 23, mail der Port 25 und WWW der Port 80 zugeordnet. Auf dem Rechner laufen dann Programme, sog. Dämonen, die den entsprechenden Port abhören. Diese Dämonen sind die Server des entsprechenden Dienstes. Auch die Installation eines Superservers (inetd) ist möglich. Dieser hört alle Ports ab. Sobald eine Nachricht auf einem Port ankommt, startet er den zugehörigen Server.

User Datagram Protocol (UDP)

Auch das UDP ist ein Potokoll der Transportschicht. Im Gegensatz zum TCP haben aber hier Nachrichten Paketgröße und bei der Übertragung können Pakete verloren gehen oder in falscher Reihenfolge ankommen. Der Benutzer muß gegebenenfalls selbst die Kontrolle übernehmen und verlorene Pakete explizit neu anfordern. Trotz dieser Nachteile wird UDP häufig eingesetzt, da es schnellere Datenübertragung ermöglicht und wenn man den Verlust einiger Daten verkraften kann.

Multicast IP

Ein Teil der für spätere Anwendungen reservierten IP-Adressen werden neuerdings als Adressen für sogenannte Multicast-Gruppen verwendet. Rechner können einer Gruppe beitreten und erhalten dann alle Nachrichten, die an eine Gruppe geschickt wird. Technisch funktioniert Multicast IP über ein virtuelles Teilnetz im Internet. Dieses Teilnetz wird auch als Mbone bezeichnet. Informationen zum Mbone findet man auf der Webseite www.mbone.com. Derzeit kann man nur per UDP Nachrichten an solche Gruppen schicken, d.h. IP Multicast ist ein unzuverlässiges Protokoll. Es sind derzeit zwar auch einge zuverlässige Protokolle vorgeschlagen, aber dise sind meist anwendungsabhängig. Das Mbone ist ein Netz im Netz. Es besteht aus Multicast Routern, die über TCP/IP miteinander kommunizieren. Außerdem schicken die Multicast Router auch Nachrichten zu den Rechern, die bei ihm direkt für eine Gruppe eingetragen sind.



Multicast verhindert Engpässe (Bottlenecks) durch Lastenverteilung. Ohne Multicast müßte ein Rechner N Nachrichten an alle N Mitglieder einer Gruppe schicken. Mit Multicast sende er eine Nachricht an den nächsten Multicast Router. Dieser sendet k Nachrichten an die k nächsten Multicast Router und j Nachrichten an die direkt bei ihm eingetragenen Gruppenmitglieder. In der Regel sind k und j sehr viel kleiner als N. Mit anderen Worten, die Nachricht wird so nahe wie möglich als einzelne Nachricht zu den

Rechnern der Gruppe transportiert, erst der letzte Multicast Router vor dem Ziel, erzeugt dann mehrere Exemplare dieser Nachricht.

Architekturen für verteilte Welten

Die meisten verteilten, virtuellen Welten verwenden derzeit Client/Server-Architekturen. Hierbei verwaltet ein Server der aktuelle Zustand der Welt . Die Clienten schicken alle Änderungen an den Server und dieser propagiert sie zu allen anderen Clienten. Da alle Kommunikation über den Server läuft, stellt er den Engpaß (Bottleneck) dar. Auch vom Standpunkt der Fehlertoleranz ist diese Lösung unausgereift, da das ganze System ausfällt, sobald der Server ausfällt (single-point-of-failure). Ein anderes Problem besteht darin, wenn ein Client Daten cached oder über einen Proxy, der die Daten cached, mit dem Server verbundenist. In diesem Fall müssen zusätzlich komplexe Cache-Konsistenzprotokolle verwendet werden.

In Peer-to-Peer Architekturen hat jeder Client eine Teilkopie der Daten. Der aktuelle Zustand der Welt ist auf alle Clienten verteilt. Bei Änderungen muß ein Client diese an alle anderen Clienten schicken. Fällt ein Client aus, gehen seine Teildaten verloren, aber die restlichen Clienten können ohne ihn weiter arbeiten.

Client/Server und Peer-to-Peer Architekturen sind die Extreme in einem weiten Spektrum an möglichen Architekturen. Für große, flexible, verteilte, virtuelle Welten verwendet man hybride Architekturen, d.h. Vereinigungen beider Modelle. Den Server benötigt man zum initialen Laden der Welt und zur Speicherung des aktuellen Zustands der gesamten Welt oder ihrer Teile (Persistenz). Peer-to-Peer Kommunikation bietet bessere Performanz und geringere Ausfallzeiten.

Konsistenz

Die wichtigste Aufgabe von Protokollen für verteilte, virtuelle Welten besteht darin, die Daten auf den beteiligten Rechnern konsistent zu halten. Dazu genügt es nicht, die Änderungen auf einem Rechner zu allen anderen zu übertragen, da gleichzeitig auf mehreren Rechnern widersprüchliche Änderungen (datenbezogene Inkonsistenz) stattfinden können und die Kommunikation über das Netzwerk eine nicht vernachlässigbare Verzögerung mit sich bringt (network latency). Dadurch kann sich die zeitliche Reihenfolge, in der die Änderungen bei verschiedenen Rechnern eintreffen, unterschiedlich sein. Die Änderungen sollten aber auf allen Rechnern in der gleichen Reihenfolge stattfinden (zeitliche Konsistenz). Protokolle unterscheiden sich darin, für welche Daten sie welche Konsistenz garantieren. Bei strikter Konsistenz müssen alle Änderungen zuverläsig an alle Clienten geschickt und Konflikte vermieden oder aufgelöst werden. Im Internet setzen solche Protokolle meist direkt auf TCP auf. Im Gegensatz hierzu werden bei schwacher Konsistenz (best effort) alle Änderungen, so gut es geht, an alle Clienten geschickt. Im Internet setzen solche Protokolle meist auf UDP auf. Bei großen, verteilten, virtuellen Welten versucht man den Aufwand für die Konsistenzhaltung zu reduzieren. Dazu gibt es zwei Ansätze. Zum einen weist man bestimmten Daten bestimmte Konsistenzgruppen (Teilmengen aller Clients) zu, d.h. die Änderungen dieser Daten müssen nur an die Mitglieder dieser Gruppen geschickt werden. Zum anderen kann man auch den Konsistenzbegriff abschwächen (relaxation). Die zeitliche Konsistenz kann man für Daten aufheben, bei denen die zeitliche Abfolge der Änderungen nicht bei allen Clienten gleich sein muß. Die datenbezogene Konsistenz kann man abschwächen, indem man bei widersprüchlichen Werten den Mittelwert bildet oder Abweichungen bis zu einem gewissen Grad erlaubt (siehe dead reckoning).

Protokolle für verteilte, virtuelle Welten

Es gibt noch nicht **den** Standard, aber man kann von den vorgeschlagenen Standards viel lernen.

Distributed Interactive Simulation DIS

1985 startet das amerikanische Verteidigungsminisiterium ein Projekt namens SIMNET zur Entwicklung einer Computersimulation eines Schlachtfelds mit Panzern, Flugzeugen, Geschossen, etc. . Dabei sollte die Berechnung der Simulation auf viele Rechner im Netz verteilt sein. Eine Vielzahl der an der Simulation beteiligten Objekte sollten interakiv sein, d.h. daß sie von Menschen während der Simulation gesteuert wurden. Die Schlachtfeldsimulation sollte zur Ausbildung von Panzerfahren bishin zu Kommandeuren eingesetzt werden. Als Nachfolgeprojekt wurde 1989 DIS gestartet und das gleichnamige Protokoll wurde 1993 IEEE-Standard. Derzeit läuft eine Nachfolgeprojekt namens HLA (high-level architecture, ähnlich CORBA). Trotz seines beschränkten und vielleicht auch fragwürdigen Anwendungsgebiets hat DIS einige Begriffe und Ideen geprägt, die auch spätere zivile Protokolle beeinflußt haben. In DIS wurden Nachrichtenformate festgelegt, sogenannte PDUs (protocol data unit). Im Standard gibt es 27 solcher PDUs für so Die wichtigste PDU ist jedoch die Entity State PDU. In ihr wird der gesamte Zustand eines Objekts übertragen, so enthält sie u.a. dessen Position, Ausrichtung und Geschwindigkeit.

Header
Entity ID
…
EntityType
…
Linear Velocity
Location
Orientation
Appearance
Dead Reckoning Parameters
…

Da immer der gesamte Zustand eines Objekts übertragen wird, braucht der Empfänger nicht alle vorangegangen Objektzustände empfangen zu haben (stateless). Außerdem verlangt DIS von jedem Objekt, daß es regelmäßig seinen Zustand an alle anderen schickt (heartbeat). DIS ist ein Peer-to-Peer Protokoll, aber wegen des Heartbeats ist es für eine Simulation mit vielen Objekten ungeeignet, es skaliert nicht. Nach einer Schätzung erfordert DIS für 100.000 Teilnehmer zu jedem dieser Teilnehmer eine

Verbindung mit einer Bandbreite von 375 Mbit/s (siehe Dissertaton von M. Macedonia, 1995).

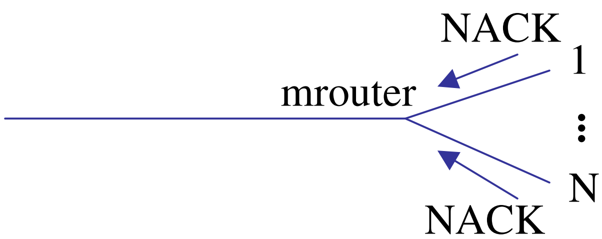
Distributed Worlds Transfer and Communication Protocol DWTP

DWTP wurde bei der Gesellschaft für Mathematik und Datenverarbeitung (GMD) von Wolfgang Broll entwickelt. Es ist ein Anwendungsschichtsprotokoll für große, verteilte, virtuelle Welten. Im Gegensatz zu DIS ist DWTP anwendungsunabhängig und heterogen, d.h., es werden verschiedene Datentypen (Events, Nachrichten, Dateien und kontiniuerliche Ströme) übertragen. Der Hauptaugenmerk wurde jedoch bei der Entwicklung von DWTP auf Skalierbarkeit gelegt. In DWTP gibt es zwei Rollen: Teilnehmer und Dämonen. Ein Dämon kann gleichzeitig für mehrere Welten zuständig sein. Skalierbarkeit zum einen durch die Verwendung von Multicasting und zum anderen dadurch, daß Dämonen auf verschiedenen Rechnern laufen können, aber nicht müssen. DWTP unterscheidet folgende Arten von Dämonen:

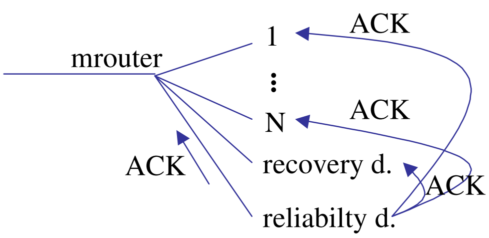
- Der World-Dämon überträgt die gesamte aktulelle Welt an neue Teilnehmer.
- Der Reliability-Dämon entdeckt Übertragungsfehler.
- Der Recovery-Dämon bietet Unicast-Verbindung zum Übertragen verlorener Pakete.
- Der Unicast-Dämon erlaubt Verbindungen mit nicht multicast-fähigen Teilnehmern.

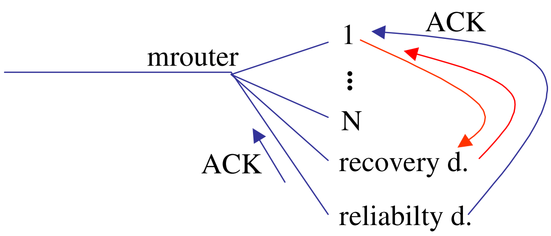
Im Folgenden wird erläutert, wie diese Dämonen bei der Fehlerbehandlung in DWTP zusammenarbeiten. Da IP Multicasting auf UDP aufsetzt, ist die Übertragung unzuverlässig, d.h. es können Pakete verloren gehen. Um Zuverlässigkeit zu gewährleisten, muß man daher in der Anwendungsschicht, also hier in DWTP, solche Fehler finden und beheben. Warum scheinbar komplexe Architektur in DWTP gewählt wurde, sieht man besser ein, wenn man zuvor betrachtet, wie Fehlerbehandlung traditionell, oder besser gesagt adhoc, betrieben wird.

Viele Protokolle (z.B. ISTP, RAMP, RMP) verwenden NACKs (negative acknowledge message), d.h. die Clients schicken, falls sie ein Paket nicht erhalten haben, eine Nachricht an den Sender. Da der Übertragungsfehler aber häufig durch einen überlasteten Multicast Router verursacht wurde, wird für diesen durch die zusätzlichen NACKs die Belastung noch größer. Manche Protokolle lösen dieses Problem, dadurch daß die Clienten eine zufällig lange Zeit warten, bevor sie die NACK Nachricht schicken. (Eine ähnliche Strategie wird im Ethernet verwendet, um Konflikte zu lösen, wenn mehrere Rechner gleichzeitig auf den Bus zugreifen wollen.)



In DWTP werden die Nachrichten nicht im Fehlerfall an den Sender geschickt, sondern nur einer oder einige der Empfänger, nämlich die Reliability-Dämonen, schicken mit der Paketnummer versehene ACK Nachrichten (acknowledge message) an den Sender und an alle Teilnehmer, für die sie zuständig sind. Ein Teilnehmer erkennt ein Problem, wenn er ein ACK für ein Paket erhält, aber nicht das Paket selbst. In diesem Falle fordert er bei einem Recovery-Dämon, das Paket erneut an. Dieser sendet das Paket dann über zuverlässiges Unicast (also z.B. TCP) an den Teilnehmer. Falls der Recovery-Dämon das Paket auch nicht erhalten hat, muß die gesamte, aktuelle Welt von einem World-Dämon neu geladen werden.





Virtual Reality Transfer Protocol VRTP

Bei VRTP handelt es sich um ein laufendes Projekt, d.h. es gibt noch keine fertige Implementierung oder Spezifikation. Dennoch ist es schon jetzt interessant, zu sehen, welche Ziele sich die Entwickler gesetzt haben. Auch in VRTP kann ein Rechner mehrere Rollen erfüllen, nämlich die eines Clienten, eines Servers oder eines gleichberechtigten

Teilnehmers, also Peer. Die Aufgabe des Clients besteht darin, eine Welt anzuzeigen, also z.B. ein VRML-Browser. Ein Server verwaltet eine Welt und sendet sie an andere, z.B. über das Hypertext Transfer Protocol HTTP. Wesentliche Bestandteile der Architektur von VRTP sind Peer-to-Peer Kommunikation und Multicasting. Auch in VRTP hat man erkannt, daß die Fehlerbehandlung eine große Rolle spielt, will man ein skalierbares und brauchbares Protokoll für verteilte, virtuelle Welten entwickeln. Deshalb sollen Netzwerküberwachungsmechanismen (network monitoring) in VRTP integriert werden, um Übertragungsprobleme zu erkennen und zu lösen. Im Gegensatz zu DWTP soll VRTP eigenständig das für den Anwendungsfall geeignete Netzwerkprotokoll wählen.

Die Entwickler von VRTP beabsichtigen, daß VRTP für VRML-basierte verteilte, virtuelle Welten, die gleiche Rolle spielen soll, wie HTTP für HTML. Diesen Anspruch kann man derzeit wohl noch für übertrieben halten, interessant ist aber die Einsicht die dahinter steckt, nämlich, daß der Erfolg des WWW darin begründet ist, daß man existierende Protokolle und Techniken auf intelligente Art miteinander kombiniert und nicht versucht, das Rad neu zu erfinden. Im Falle von VRTP sollen diese Protokolle und Techniken HTTP, DIS, multicast streaming, Java Agenten und Netzwerkmonitoring sein.

Wie DWTP ist auch VRTP ein heterogenes Protokoll, d.h. es kennt unterschiedliche Arten von Daten mit unterschiedlichen Übertragungsanforderungen:

- Leight-Weight Interactions: Dies sind Ereignisse, Zustands- und Kontrollinformationen, einschließlich PDUs. Sie erfordern nur unzuverlässige Übertragung.

- Network Pointer: Hierbei handelt es sich um Referenzen auf Objekte in einem globalen Adreßraum, also z.B. URLs.

- Heavy-Weight Objects: Hiermit sind Dateien gemeint, die zuverlässig übertragen werden müssen.

- Real-Time Streams: Dies sind kontiniuerliche Daten, etwa Live Video, Live Audio, oder Verhaltensströme, die über Multicast unzuverlässig, aber schnell übertragen werden müssen.

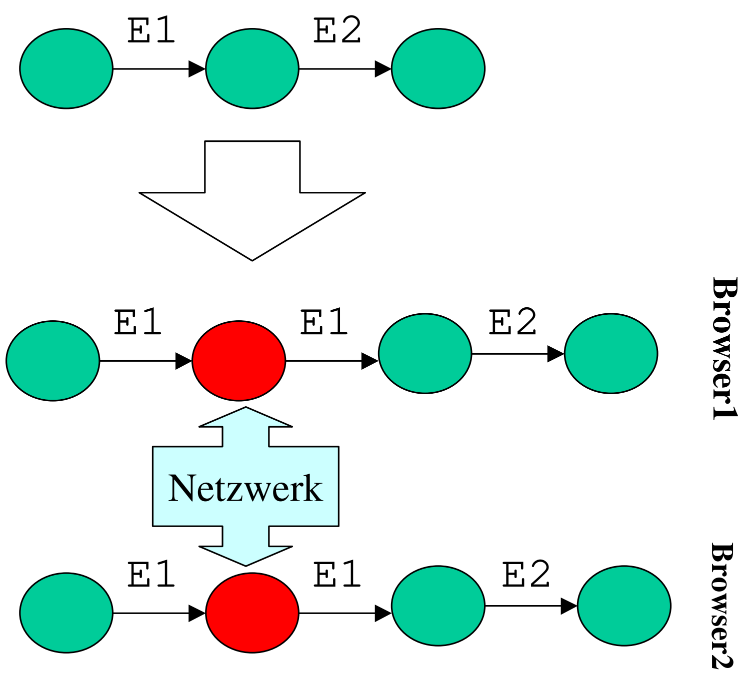
Die wohl interessanteste Idee im Zusammenhang mit VRTP ist aber das sogenannte dial-a-behavior-Protokoll DBP, ein Metaprotokoll, mit dem sich Form und Inhalt der Datenpakete eines Protokolls beschreiben lassen und das es ermöglicht zur Laufzeit neue Protokolle einzuführen, ein Objekt kann also z.B. anderen Objekten mitteilen, welche neue Nachrichten es versteht und welche neuen Nachrichten es an die anderen Objekte schickt und wie diese diese Nachrichten zu interpretieren haben. Einen Interpreter für die neuen Nachrichten könnte das Objekt z.B. in Form einer Java-Klasse an die anderen Objekte schicken.

VRML Erweiterungen für verteilte Welten

In diesem Abschnitt werden zwei Erweiterungen von VRML vorgestellt, d.h. neue Knotentypen, mit denen verteilte Szenen implementiert werden können. Zuerst wird VSPLUS vorgestellt, das es erlaubt, Events über einen Zwischenknoten an alle anderen Browser zu übertragen. Dann wird Living Worlds vorgestellt, ein Standard, der unter Beteiligung der führenden Hersteller in diesem Bereich ausgearbeitet wurde. Das Abstraktionsniveau von Living Worlds ist recht hoch und erlaubt den Herstellern weiterhin den Einsatz ihrer propritären Lösungen (Protokolle und APIs) auf niedrigerem Niveau. Sony verwendet z.B. VSPLUS zur Implementierung von Living Worlds.

VSPLUS: Wie macht man aus Einbenutzerwelten auf einfache Weise Mehrbenutzerwelten?

Ausgangspunkt für die Entwicklung von VSPLUS war die Idee, daß es eine große Zahl von VRML-Szenen gibt, die für den Einbenutzerbetrieb entwickelt wurden und daß es



einen großen Bedarf geben wird, solche Szenen auf einfache Weise mehrbenutzerfähig zu machen. Zu diesem Zweck fügt VSPLUS zu VRML sogenannte Netzknoten (net nodes) hinzu. Diese Knoten werden verwendet, um Events an Instanzen der VRML-Szene, die augenblicklich in anderen Browser im Netzwerk geladen sind, weiterzuleiten. Ein solcher Event wird auch als verteilter Event bezeichnet.

Die Synchronisation und Übertragung der Daten zwischen den Netzknoten übernimmt die Konsistenzmanager (consistency manager). Für jeden Netzknoten gibt es zur Laufzeit einen zuständigen Konsistenzmanager im Netzwerk, alle anderen Instanzen des Netzknotens auf anderen Rechnern, senden ihre Events über das Netzwerk an diesen.

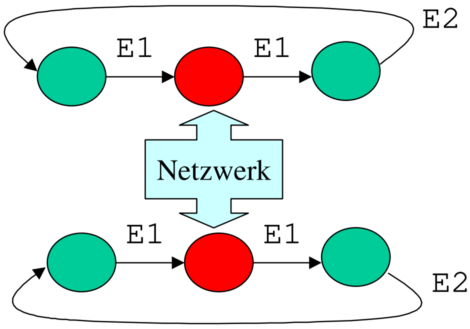
Für Avatare, d.h. benutzergesteuerte Objekte, ist der Konsistenzmanager auf dem Rechner des Benutzers, für programmgesteuerte Objekte, sogenannte Bots, auf dem Rechner auf dem das Programm läuft. Die Aufgabe des Konsistenzmanagers besteht darin, Konflikte zu erkennen und aufzulösen. Ein Konflikt entsteht, wenn der Konsistenzmanager von mehreren Teilnehmern gleichzeitig verschiedene Werte für einen Event empfängt. Der Konsistenzmanager kann diesen Konflikt durch mathematische Operationen, etwa Mittelwertbildung, auflösen, oder er kann ein Abstimmen (voting) initiieren, d.h. die Sender der Events entscheiden per Mehrheitsentschluß, welcher Wert genommen wird. Zusätzlich bietet der Konsistenzmanager die Möglichkeit der Konfliktvermeidung durch Zugriffskontrolle, d.h. bevor ein Netzknoten einen Event schickt, fordert er beim Konsistenzmanager dazu das Recht an. Nach dem Senden des Events wird das Zugriffsrecht wieder abgegeben. Diese Art der Zugriffskontrolle wird durch eine Semaphore geregelt. Hat die Semaphore den Wert 0, kann kein weitere Netzknoten ein Zugriffsrecht erhalten. Auf diese Weise kann der Konsistenzmanager Konfliktauflösung und Zugriffskontrolle für einen Event kombinieren. Zum Beispiel könnte er für einen numerischen Event, die Semaphore mit 2 initialisieren. Ein erster Netzknoten fragt nach dem Zugriffsrecht und erhält es, die Semaphore wird auf 1 erniedrigt. Ein zweiter Netzknoten fragt nach dem Zugriffsrecht und erhält es, die Semaphore wird auf 0 gesetzt. Einem dritten Netzknoten wird das Zugriffsrecht verweigert. Die beiden anderen Senden ihre Werte für den Event und der Konsistenzmanager bildet den Mittelwert und schickt ihn weiter, die Semaphore wird wieder auf 2 gesetzt.

Für den Fall, daß ein Browser (late comer) eine Szene erst lädt, nachdem schon andere Browser diese geladen haben und sich der Zustand der Szene geändert hat, sieht VSPLUS folgende Strategie vor: Ein neu hinzu gekommener Browser muß seine Szene initialisieren, indem er beim Konfliktmanager für jeden verteilten Event des letzten Wert anfordert. Erst nach dieser Initialisierung darf der Browser selbst Events auslösen. Auch den Fall, daß das Netzwerk zeitweise nicht erreichbar ist, hat man in VSPLUS bedacht. Der Browser soll dann im Einbenutzerbetrieb arbeiten, d.h. falls die Netzwerkverbindung unterbrochen wurde, lassen die Netzknoten die verteilten Events einfach durch, ohne sie an den Konsistenzmanager weiterzuleiten.

Eine der interessantesten Eigenschaften von VSPLUS ist der Umgang mit **Eventströmen**. Ihm liegt die Einsicht zugrunde, daß auf einem Browser kontinuierliche Events, etwa die Takte einer Uhr (TimeSensor) oder die Positionsänderungen der Maus (DragSensor), wesentlich schneller erfolgen und abgearbeitet werden können, als es die Übertragung über ein Netzwerk erlauben würde. Daher bietet VSPLUS für solche Events die Möglichkeit, deren Werte zu ‘samplen”, d.h. nur nach festgelegten Intervallen den zuletzt aufgetretenen Wert dieses Events zu übertragen.

In VRML-Browsern werden Event-Loops, d.h. daß ein Event sich selbst direkt oder indirekt wieder auslöst, aufgebrochen. Dies ist notwendig, da der Browser zwischen zwei Frames, die er auf dem Bidlschirm ausgibt, alle Events, die nach dem Anzeigen des ersten Frames aufgetreten sind, abgearbeitet haben muß. Da alle Events die zum gleichen Zeitpunkt auftreten, den gleichen Zeitstempel erhalten und Events, die durch andere ausgelöst werden, den gleichen Zeitstempel wie der sie auslösende Event erhalten, würde der Browser immer neue Events auslösen, ohne je das nächste Frame zeichnen zu können. Daher werden Schleifen wie folgt aufgebrochen: Wird ein Event A ein durch einen anderen Event B ausgelöst, der den gleichen Zeitstempel hat, wie er A bei seiner letzten Aktivierung zugewiesen wurde, dann wird die Schleife an dieser Stelle aufgebrochen, indem der neu ausgelöste Event A erst nach dem Zeichnen des nächsten Frames bearbeitet wird und dann einen späteren Zeitstempel erhält. Ein VRML-Browser kann eine Event-Loop auch einige Male durchlaufen, bevor er sie aufbricht, entscheidend ist, daß er anhand der Zeitstempel solche Schleifen zur Laufzeit erkennen kann.

Aufgabe des Konfliktmanagers ist es auch, die Kausalität, d.h. welcher Event welchen auslöst, so zu erhalten, wie er im Einbenutzerbetrieb vorlag. Da aber die Zeitstempel der Events nur lokal gültig sind, bereiten Event-Loops, die Netzknoten enthalten, in VSPLUS Probleme.

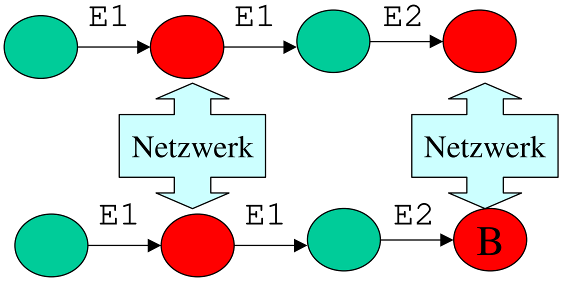


In Abbildung ?? ist eine solche Schleife dargestellt. Das Problem ergibt sich dadurch, daß die Zeitstempel nicht mitübertragen werden, da sie auf einem anderen Rechner keinen Sinn haben, dieser hat ja eine eigene Uhr. Es wäre ein Protokoll zum Synchronisieren der Uhren auf allen Browsern notwendig. Um den Problemfall deutlich zu machen, führen wir folgende Notation (e1,b1,t1) -> (e2,b2,t2) ein. Sie besagt: Event e1 mit Zeitstempel t1 auf Browser b1 löst Event e2 mit Zeitstempel t2 auf Browser b2 aus. Dann könnte folgende Aktivierungskette auftreten:

(E1,1,t1) -> (E1,2,t2) -> (E2,2,t2) -> (E1,2,t2) -> (E1,1,t3) -> (E2,1,t3) -> (E1,1,t3)

Im Gegensatz zum Einbenutzerbetrieb kann nun der Browser am Zeitstempel nicht mehr erkennen, daß (E1,1,t3) durch (E1,1,t1) ausgelöst wurde und folglich eine Event-Loop vorliegt.

Ein weiterer Problemfall besteht in einem Szenario, daß die Entwickler von VSPLUS mit Message-Expansion bezeichnen. In Abbildung ?? wird dieser Fall dargestellt.



CM in B erhält E2 zweimal

Der mit B bezeichnet Netzknoten könnte gleichzeitig zwei E2-Events erhalten, einen von einem lokalen Knoten und einen über das Netzwerk. Der Konsistenzmanager für B kann nun nicht entscheiden, ob diese beiden Events durch den gleichen ursprünglichen Event ausgelöst wurden, d.h. es handelt sich eigentlich um den gleichen Event E2, oder ob sie durch unterschiedliche Events ausgelöst wurden, etwa dadurch, daß E1 auf Browser 1 zweimal nacheinander ausgelöst wurde.

Die beiden Aktivierungsketten für den ersten Fall:

(E1,t1,1)->(E2,t1,1)->(E2,t2,2)   (E1,t1,1)->(E1,t3,2)->(E2,t3,2)

Die beiden Aktivierungsketten für den zweiten Fall:

(E1,t1,1)->(E2,t1,1)->(E2,t2,2)   (E1,t3,1)->(E2,t3,1)->(E2,t4,2)

In beiden Fällen sind die Zeitstempel verschieden, die die in B ankommenden Events haben und können daher nicht verwendet werden, um festzustellen, daß es sich um die gleichen, bzw. um zwei verschiedene Events handelt.

Beide Problemfälle, Event-Loops und Message-Expansion, sind in VSPLUS verboten und es existieren Werkzeuge, die VRML-Dateien mit Netzknoten statisch daraufhin untersuchen, ob solche Fälle vorliegen.

Living Worlds – der kommende Standard für verteilte Welten?

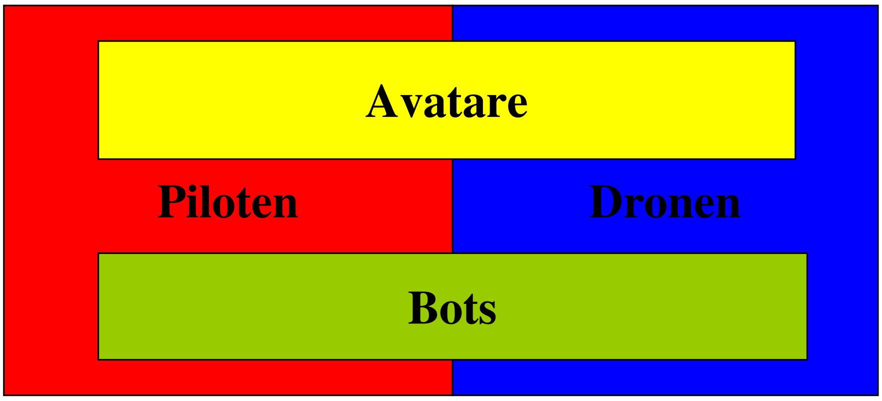
Eine Arbeitsgruppe des VRML-Konsortiums erarbeitet derzeit einen Standard für verteilte, virtuelle Welten, der auf VRML beruht. Mit VRML 1.0 wurde ein Standard für statische Szenen geschaffen, das heißt man kann in VRML 1.0 das Aussehen von Objekten festlegen. Als nächster Schritt wurde mit VRML 2.0 die Möglichkeit gegeben, Verhalten von Objekten zu spezifizieren. Der zukünftige Standard soll nun auch das Spezifizieren verteilter Szenen ermöglichen, d.h. mehrere Benutzer sollen in auf verschiedenen Rechnern laufenden Kopien **einer** Szene interagieren können. Dem Abstraktionsniveau von LivingWorlds wurde so gewählt, daß die Eigenschaften der Kommunikation zwischen den Kopien der Szene und der darin enthaltenen Objekte deklariert werden kann und nicht programmiert werden muß. Es wird sogar von der Kommunikationsarchitektur abstrahiert, d.h. es wird nicht festgelegt, ob die Kommunikation zwischen den verschiedenen Browsern, die eine Szene geladen haben, über ein Client/Server-Protokoll oder ein Peer-to-Peer-Protokoll erfolgt. Die Entwickler von LivingWorlds bezweifeln, daß die Zeit reif ist für ein standardisiertes Wire-Protokoll (etwa DWTP oder VRTP), da es noch nicht genügend Erfahrungen aus Anwendungen gibt und die Optimalitätsanforderungen teilsweise widersprüchlich sind. Schaut man sich die Schlüsselfiguren in der LivingWorlds-Arbeitsgruppe (Sony, Blaxxun, HP und Paragraph) an, so liegt die Vermutung nahe, daß die Motivation für das gewählte, hohe Abstraktionsniveau darin liegt, die Investitionen in die eigenen, proprietären Wire-Protokolle und APIs zu retten. Warum sollte man dann aber überhaupt Interesse an einem Standard haben. Die Motivation für die Industrie liegt darin, die Interoperabilität verschiedener Komponenten zu ermöglichen. Solche Komponenten können Avatare, sonstige Objekte, die Merhbenutzertechnologie oder ganze Teilwelten sein. Interoperabilität wird durch festgelegte, möglichst minimale Schnittstellen für Komponenten erreicht. Die Vorteile sind vielfältig:

- Benutzer können Objekte und Avataren in anderen Welten wiederverwenden.

- Entwickler von Komponenten haben größeren Kundenkreis.
- Entwickler von Anwendungen können Komponenten verschiedener Anbieter kombinieren.

Interessant an LivingWorlds ist die Festlegung einer Terminologie. Dadurch werden überladene Begriffe eingegrenzt und Mißverständnisse vermieden. Zuerst kann man **Objekte und Objektinstanzen** unterscheiden. Für jedes Objekt einer Szene existieren auf jedem Browser, der die Szenen geladen hat, genau eine Objektinstanz. Eine Objektinstanz ist der Teil des Szenengraphen im Browser, der dieses Objekt repräsentiert.

Objekte, deren Objektinstanzen unabhängig sind, d.h. deren Objektinstanzen nicht untereinander konsistent gehalten werden müssen, werden als **lokale Objekte** bezeichnet. Solche deren Objektinstanzen konsistent sein müssen, nennt man **verteilte Objekte**. Bei den Instanzen verteilter Objekte kann man weiter zwischen Piloten und Dronen unterscheiden. Ein **Pilot** ist eine Objektinstanz, die eigenständig den Zustand des Objekts ändert. Eine **Drone** ist eine Objektinstanz, die Zustandsänderungen des Objekts mitvollzieht. Verteilte Objekte kann man weiterhin nach Kontrolle ihres Piloten unterteilen. Ein Avatar ist ein verteiltes Objekt, dessen Pilot von einem Menschen gesteuert wird. Ein Bot (von robot) ist ein verteiltes Objekt, dessen Pilot von einem Programm gesteuert wird.



Eine Instanz eines verteilten Objekts kann während seiner Lebenszeit seinen Status verändern, d.h. ein Pilot kann eine Drone werden und umgekehrt, ein Bot kann ein Avatar werden und umgekehrt.

Eine **Szene** wird als Menge von Objekten, die in einem dreidimensionalen Raum angeordnet sind, definiert. Eine **Welt** ist dann eine durch Links (Anker) verbundene Menge von Szenen und eine **Zone** ist ein begrenzter Teil einer Szene, der verteilte Objekte enthält.

In jeder Zone sorgt eine Komponente, die die Schnittstelle zwischen Browser und Netzwerk bereitstellt, für die Konsistenzhaltung und Kommunikation zwischen den Instanzen der verteilten Objekte. Diese Komponente wird **MUTech** (multi-user technology) genannt.

Beispiel für Zonendefinition

```
DEF Z Zone {
        private DEF PZ PrivateZone {
```

```
        zone USE Z
        muTech SomeCompanyZone { private USE PZ
                                 … }
        whichTechnology ”SomeCompany"
    }
}

DEF PROX ProximitySensor{ size 100 100 100 }

ROUTE PROX.isActive TO PZ.set_isActive
ROUTE PROX.position_changed TO PZ.set_avatarPosition
ROUTE PROX.orientation_changed TO PZ.set_avatarOrientation
```

Beispiel für ein verteiltes Objekt

```
DEF SO SharedObject {
    private DEF PSO PrivateSharedObject {
        visualDefinition
            [ Transform {
                children [
                    Shape {
                        geometry Cone{}
                    }
                ]
               } ]
        sharedObject USE SO
    }
}
```

Beispiel für Nachrichten und Selektoren

```
DEF SC Script { directOutput TRUE
                eventIn SFNode message
                eventOut SFColor setColor
                field SFColor blue 0 0 1
                field SFColor red 1 0 0
                url “javascript: function message(val,t)
                      { if (val->what=‘Blue’) setColor=blue;
                        if (val->what=‘Red’) setColor=red; }”
              }

DEF SO SharedObject {
    private DEF PSO PrivateSharedObject {
      visualDefinition [
         DEF PERSON Transform { children
            [ Shape { geometry Cone { }
                       appearance Appearance {
                         material DEF M Material {
                          diffuseColor 1.0 0.0 0.0 } } } ] } ]
      pilotSelector
        DEF SEL Selector {
            selectorItems [
                SelectorItem{ label "Body Red !!"
                              messageName "Red"
                              messageScript },
                SelectorItem{ label "Body Blue !!"
```

```
                                messageName "Blue"
                                messageScript USE SC } ] }
      states [ DEF BODYCOLOR
                   NetworkSFColor{ tag "BodyColor" } ]
      sharedObject USE SO
    }

ROUTE SC.setColor TO BODYCOLOR.set_value
ROUTE BODYCOLOR.value_changed TO M.diffuseColor
```

Kommerzielle und experimentelle Mehrbenutzerwelten

Installations- und Bedienungshinweise, sowie Anekdoten zu den folgenden System findet der Leser in Bruce Damer's Buch *Avatars!*. Bruce Damer verwendet den Begriff virtuelle Welten und grenzt ihn deutlich von virtueller Realität ab. Bei virtueller Realität geht es darum, die Wirklichkeit möglichst getreu abzubilden und dazu wird meist teuere Hardware verwendet, z.B. Cyberhelme und –anzüge. Virtuelle Welten hingegen setzen nur geringe Anforderungen an die Hardware des Benutzers, heutige PCs genügen. Die Immersivität, d.h. der subjektive Eindruck oder die Illusion, sich tatsächlich in der virtuellen Welt zu befinden, wird durch die Kommunikation mit anderen Menschen erreicht und nicht durch die ausgefeilte Hard- und Software.

Produktname	URL	Darstellung der Welt	Darstellung der Avatare
WorldsChat	www.worlds.net	3D	2D
The Palace	www.thepalace.com	2D	2D
Active Worlds	www.activeworlds.com	3D	3D
Worlds Away	www.worldsaway.com	2D	2D
Onlive! Traveler	www.onlive.com	3D	3D
Virtual Places	www.vplaces.com/vpnet	2D	2D
Black Sun's Passport	www.blacksun.com	3D	3D
Microsoft's Comic Chat	www.microsoft.com/ie/chat	2D	2D
Oz Virtual	www.oz.com/zipvrml	3D	3D
Sony's Community Place	vs.spiw.com/vs	3D	3D
Microsoft's V-Chat	Vchat1.mircosoft.com	2D & 3D	2D
NTT's Interspace	Www.ntts.com/interspace	3D	2D (Video)

Weitere experimentelle 3D Welten existieren bereits oder sind am Enstehen auf den folgenden Seiten:

Intel's IDMOO connectedpc.com/iaweb/idmoo Chaco's Pueblo www.chaco.com, SenseMedia's The Sprawl www.sensemedia.net und La Deuxième Monde www.2nd-world.fr .

Zur Darstellung der Welt verwenden einige Systeme feste Hintergrundbilder und alle Objekte und Avatare befinden sich in einer Ebene (2D), manche unterscheiden auch mehrere Ebenen (2 ½ D), d.h. Objekte können vor anderen plaziert werden und eine wachsende Zahl an Systemen verwendet dreidimensionale, modellbasierte Darstellungen. Das erste dreidimensionale System war Worlds Chat (1995). In einigen dieser virtuellen Welten können die Benutzer die Welt ändern, d.h. ganze Städte neu aus dem virtuellen Boden stampfen.

Auch bei der Darstellung der Avatare gibt es große Unterschiede. Die einfachsten Systeme verwenden ein festes Bild, manche erlauben eine Reihe von Bildern zur Darstellung animierter Bewegungen. Bei der Verwendung von 2D Avataren in 3D Welten verwenden manche Systeme mehre Ansichten für den Avatar, also z.B. von vorne, hinten und von der Seite. In Interspace werden die Avatare als kleine Bildschirme dargestellt, auf denen eine Videoübertragung von einer beim Benutzer installierten Kamera erscheint. Zusätzlich erlaubt Interspace außer dem Textchat auch die Übertragung gesprochener Sprache (Voicechat), so daß man, wie in einer Videokonferenz, den Avatar sprechen hört und sieht. Bei den meisten Systemen mit 3D-Avataren können diese auch Animationen ausführen.

In den größten dieser Welten haben im Laufe der Zeit mehrere hunderttausend Benutzer ihre Spuren hinterlassen und zu manchen Zeiten sind mehrere tausend Benutzer gleichzeitig aktiv.

Auch die Spieleindustrie bietet Mehrbenutzerspiele an, die viele Aspekte von verteilten Mehrbenutzerwelten erfüllen: Meridian59 meridian.3do.com/meridian (3D), The Realm www.realmserver.com (2 ½D), Ultima Online www.owo.com (3D), Diablo www.blizzard.com (3D), War Birds www.icigames.com (3D), Quake www.idsoftware.com (3D), Xwing vs. Tie Fighter www.lucasarts.com (3D), DukeNukem3D duke.gtinteractive.com (3D).

Außer im Spielemarkt ist der kommerzielle Erfolg von Merhbenutzerwelten noch sehr gering. Meiner Einschätzung nach liegt dies außer an den technischen Voraussetzungen, vor allem daran, daß die Welten nicht zielgerichtet eingesetzt werden, d.h. z.B. für bestimmte Arbeitsabläufe, sondern meist von Technikenthusiasten zum reinen Vergnügen besucht werden.

Navigieren in einer Mehrbenutzerwelt

Bei den kommerziellen Mehrbenutzerwelten muß man zuerst den Browser des jeweiligen Herstellers installieren. Demo-Versionen dieser Browser sind meist umsonst über das Internet ladbar. Nach der Installation startet man den Browser und muß sich zuerst einloggen. Bei manchen Systemen kann man sich als Gast anonym einloggen, andere verlangen, daß man sich registriert – meist durch Angabe der Email-Adresse. Leider wird die Anonymität manchmal dazu mißbraucht, andere Benutzer durch vulgäre Sprache oder durch andere Aktionen je nach System zu belästigen. Manche Systeme sind moderiert, d.h. es gibt einen Moderator, der sich die Dialoge anschaut und aufdringliche Benutzer vor die Tür setzt.

Nach dem Einloggen erhält man einen Default-Avatar. In Worlds Chat ist dies zum Beispiel ein Dummy, wie man ihn von Crashtests her kennt. Gefällt einem der zugewiesene Avatar nicht, kann man einen neuen aus einer Liste auswählen. In den meisten Systemen können registrierte Benutzer eigene Avatare erstellen oder importieren.

Nun kann sich der Benutzer durch die virtuelle Welt bewegen. Dabei kann er aus verschiedenen Blickwinkeln die Szenerie betrachten. Meist wird er dies aus der Sicht seines eigenen Avatars tun (first person view) oder Umschalten auf Vogelperspektive (bird's eye view). In dreidimensionalen Computerspielen hat sich die Sicht aus einer dem Avatar folgenden Kamera bewährt (third-person view), da sie dem Benutzer die Position sowie die Aktionen des eigenen Avatars am besten vor Augen führt.

Welten bestehen in der Regel aus mehreren Szenen, die auf unterschiedliche Weisen miteinander verbunden sind. Die einfachste Methode besteht in anklickbaren Objekten oder geschlossenen Türen, die ähnlich wie ein Hyperlink in einer Webseite funktionieren. Wenn man sie anklickt, wird eine neue, mit diesem Objekt verbundene Szene geladen und der Avatar des Benutzers darin plaziert. Für solche Objekte gibt es die unterschiedlichsten Bezeichnungen: anchor, hotspot, door, farcaster, wormhole, tunnel oder teleport. Bei einer raffinierteren Form solcher Türen, kann der Benutzer durch die Tür einen Teil der dahinter vorborgenen Szene schon sehen (portals), die Szene wurde aber noch nicht vollständig geladen.

Ein zentraler Bestandteil virtueller Mehrbenutzerwelten ist die Kommunikation der Benutzer untereinander. Derzeit ist die Übertragung von Textnachrichten, das sogenannte Chatten , Standard. Einige experimentelle Systeme verwenden auch Audio- oder gar Videoübertragung (etwa NTT's Interspace). Dabei wird die Sprache manchmal auch über gewöhnliche Telefonverbindungen übertragen, andere Systeme übertragen die Sprache über das Internet selbst. Die Sprachqualität der zweiten Methode ist aber bei dem heutigen Stand der Technik nur innerhalb lokaler Netze erträglich. Betrachten wir nun den Normalfall, d.h. den Textchat. In der Regel werden alle Nachrichten an alle Teilnehmer in der Szene weitergeleitet. Es sind aber auch Vieraugengespräche möglich. Der Benutzer muß dazu auf einen Flüstermodus (whisper) umschalten, in dem seine Nachrichten dann nur an den ihm räumlich nächsten oder durch den Benutzernamen ausgewählten Avatar geschickt werden. Manche Systeme sehen darüber hinaus Benutzergruppen vor, d.h. man kann dann Nachrichten direkt an eine Gruppe schicken, die Mitglieder der Gruppe können sich dabei in verschiedenen Szenen der Welt befinden. Über die rein sprachlichen Kommunikationsmittel des Textchats verwenden die Benutzer meist noch zusätzliche Gesten, um Emotionen Ausdruck zu verleihen. Im einfachsten Fall werden Emoticons im Text verwendet, etwa ein zwinkerndes ;-) oder ein schreiendes Gesicht :-O, die der Leser sicher schon aus elektronischer Post kennt. In Comic Chat kann der Benutzer durch die Form der Sprechblase und der Schriftart (fett, kursiv, etc.) das Wesen seiner Äußerungen verdeutlichen. In vielen Systemen kann man aus einem Menu den Gesichtsausdruck des eigenen Avatars oder Bewegungen (Tanzen, Hüpfen, Verbeugen) des eigenen Avatars auswählen. Da die Ausdrucksmittel noch sehr begrenzt sind, kann es auch passieren, daß ein Bot (manchmal auch als agent oder angel bezeichnet), d.h. ein programmiertes, aktives Objekt, dem Benutzer vortäuscht, ein Avatar zu sein. Die Grundlage eines solchen Bots könnte z.B. ein Nachfolger des klassischen KI-Programms Eliza sein, das Joseph Weizenbaum Anfang der 1996 Jahre am MIT entwickelt hat und das in natürlicher Sprache die Reaktionen eines Therapeuten der Rogers-Methode imitierte, einer bestimmten Form der Gesprächstherapie .

Mehrbenutzerwelten – die Zukunft?

Mehrbenutzerwelten bieten eine neue Art, mit anderen Menschen gemeinsam etwas zu unternehmen. In Mehrbenutzerwelten können Menschen neue Gemeinschaften bilden, die sich durch gemeinsame Interessen auszeichnen und die nicht durch den Ort der jeweiligen Personen in der realen Welt eingeschränkt sind. Ähnlich wie man zu den Zeiten der ersten Telefone, Radios oder Fernseher hat kaum absehen können, wie sich diese Medien einmal entwickeln, so scheitern alle Voraussagen über die Zukunft von Mehrbenutzerwelten an unserer mangelnden Fantasie. Denn mit den Technologien wandeln sich auch die sozialen Gegebenheiten, und es entstehen neue Bedürfnisse, die gerade durch diese Technologien befriedigt werden können. Im Nachhinein ist es dann auch meist unklar, ob der Bedarf die Technologie oder die Technologie den Bedarf gefördert hat.

Java-Netzwerkprogrammierung am Beispiel einer einfachen verteilten, virtuellen Welt

Einleitung

Java bietet dem Programmierer ein reichhaltiges Angebot an Protokollen zur Netzwerkkommunikation. Diese reichen von HTTP-Verbindungen, über TCP, UDP und Multicast UDP, bishin zu verteilten Objekten, die mit Hilfe von Remote Methode Invocation RMI und Object Serialisation implementiert werden können. 3D-Browser, die auf dem VRML97 Standard beruhen, bieten Schnittstellen, wie das External Authoring Interface EAI, zu benutzerprogrammierten Java-Programmen. Dadurch ist es möglich, die 3D-Szene von Java aus zu steuern und sie über Java-Netzwerkprogrammierung mit anderen Instanzen der Szene auf anderen Rechnern im Netz kommunizieren zu lassen. In diesem Kapitel wird zuerst eine Einführung in die Netzwerkprogrammierung mit Java gegeben. Anschließend werden die Architektur und einige Implementierungsdetails einer verteilten, dreidimensionalen Welt, die mit Hilfe von Java implementiert wurde, vorgestellt.

Java-Netzwerkprogrammierung

HTTP-Verbindungen (Hypertext Transfer Protocol)

Zum Laden von HTML-Dokumenten, Gif, MPEG oder Audio-Dateien, sowie zum Aufruf von CGI-Skripten bietet das Netzwerk API von Java die Klasse URL. Zum Lesen einer Seite muß man zuerst ein URL-Objekt erzeugen. Der Konstruktor erhält dabei als Argument die Internetadresse des Dokuments, das man laden will.

```
URL doc =
   new URL("http://www.gamelan.com/pages/docs/index.html");
```

Zum Lesen der Seite Lesen wird ein mit doc.openStream() ein Byte-Strom geöffnet. Für das Lesen und Schreiben von Strömen stellt das Eingabe/Ausgabe-API von Java einige Hilfsklassen zur Verfügung. Solche Klassen, deren Name mit Reader oder Writer endet, sind für Text-Ströme vorgesehen. Der Konstruktor InputStreamReader() macht aus einem Byte-Strom einen Strom von Zeichen, BufferedReader() macht aus einem Strom von einzelnen Zeichen, einen Strom von Textzeilen.

```
BufferedReader in =
 new BufferedReader(new InputStreamReader(doc.openStream()));
String line;
while ((line = in.readLine()) != null) System.out.println(line);
in.close();
```

Beim Aufrufen von CGI-Skripten muß man auch Parameter, z.B. Formulareinträge, senden und anschließend die Antwort, in der Regel eine HTML-Seite, empfangen können. Dazu öffnet man eine URLConnection, über diese kann man empfangen und senden.

```
URL script =
   new URL("http://www.gamelan.com/cgi-bin/echo");
URLConnection uc = script.openConnection();
uc.setDoOutput(true);
PrintWriter out = new PrintWriter(uc.getOutputStream());
out.println(”Eine Nachricht für Mister X”);
```

```
out.close();
BufferedReader in =
new BufferedReader(new InputStreamReader(uc.openStream()));
String line;
while ((line = in.readLine()) != null)
System.out.println(line);
in.close();
```

Die Übertragung der Ausgaben erfolgt erst nach Schließen des Eingabestroms, dann kann die Antwort empfangen werden. HTTP ist ein zustandsloses Protokoll, die Verbindung besteht immer nur für die Dauer einer Übertragung.

TCP (Transfer Control Protocol)

Das Protokoll TCP baut Punkt-zu-Punkt-Verbindungen auf, bei denen beliebig viele Nachrichten in beide Richtungen übertragen werden können. Zur Kommunikation mit einem Rechner, z.B. cscip.uni-sb.de über dessen Port 3000, erzeugt man einen Socket. Dann greift man ähnlich wie bei URL auf dessen Ein- und Ausgabeströme zu.

```
Socket s=new Socket("cscip.uni-sb.de",3000)

PrintWriter out = new PrintWriter(s.getOutputStream(),true);

BufferedReader in =
new BufferedReader(new InputStreamReader(s.getInputStream()));
…
in.close();
out.close();
```

Interessanter wird die Sache, wenn man den Programmkode für einen Server betrachtet, dieser muß Anfragen von mehreren Clients gleichzeitig beantworten können. Dazu wird ein spezieller ServerSocket geöffnet. Im folgenden Beispiel erlaubt der Server, daß Clients über seinen Port 3000 Verbindungen mit ihm aufbauen. Die Methode accept() liefert einen neuen Socket, sobald ein Client eine Anfrage über Port 3000 stellt. Der neue Socket ist an einen anderen Port gebunden, der auch dem Client mitgeteilt wird, der Client schickt nun alle weiteren Anfragen an diesen neuen Port. Da der Server die Kommunikation mit mehreren Clients gleichzeitig bewerkstelligen muß, wird für jeden Client ein Thread erzeugt. Innerhalb dieses Threads wird dann die Kommunikation über die mit dem für diesen Client erzeugten Socket verbundenen Ein- und Ausgabeströme wie oben abgewickelt.

```
ServerSocket ss = new ServerSocket(3000);
Socket client = ss.accept();

while (true)
{
  warte auf Verbindung mit accept();
  erzeuge Thread für die Verbindung zum Client über neuen Socket
}
```

UDP (User Datagram Protocol)

Ein Datagramm ist eine in sich geschlossene Nachricht, deren korrekte Übertragung nicht garantiert wird. Bei der Übertragung über UDP werden keine Ströme verwendet, sondern es wird ein 256-Byte großer Puffer angelegt und die zu übertragenden Daten werden als Bytes in diesen Puffer geschrieben.

```
DatagramSocket socket = new DatagramSocket();

byte[] buf = new byte[256];
… schreibe Daten in buf
InetAddress address = InetAddress.getByName("dings.com");
DatagramPacket packet =
new DatagramPacket(buf,buf.length,address,3000);
socket.send(packet);
```

Ebenso steht beim Empfangen von Datagrammen der Inhalt der Nachricht in einem Puffer. Die Daten werden im folgenden Beispiel in einen String umgewandelt. Es wird ein DatagramSocket auf Port 3000 geöffnet und über diesen ein Datagram empfangen.

```
DatagramSocket socket = new DatagramSocket(3000);
packet = new DatagramPacket(buf, buf.length);
socket.receive(packet);
String msg = new String(packet.getData());
```

Ein Server der Nachrichten über UDP empfängt, muß auch auf diese antworten können. Dazu greift er auf die im empfangenen Datagramm enthaltene Absenderadresse und -port zu. Diese werden dann als Zieladresse und –port verwendet.

```
InetAddress address = packet.getAddress();
int port = packet.getPort();
DatagramPacket packet =
new DatagramPacket(buf, buf.length, address, port);
socket.send(packet);
```

Am Ende der Kommunikation mit UDP sollte man nicht vergessen, den Socket wieder zu schließen, damit andere Anwendungen über den durch den Socket zuvor belegten Port kommunizieren können.

```
socket.close();
```

Multicast UDP

Die Netzwerk-API von Java unterstützt auch Multicasting über UDP. Dazu kann man Datagramme an spezielle IP-Adressen schicken, nämlich solche zwischen 224.0.0.0 und 239.255.255.255 (Class-D IP Adressen). Diese Adressen stehen für Multicast-Gruppen. Zum Senden an Gruppen braucht man also keine neuen Methoden oder Klassen.

```
InetAddress group = InetAddress.getByName("230.0.0.1");
DatagramPacket packet
= new DatagramPacket(buf, buf.length,group,3000);
socket.send(packet);
socket.close();
```

Zum Empfangen von Nachrichten an eine Gruppe muß man allerdings etwas mehr Aufwand treiben. Zuerst wird ein MulticastSocket geöffnet, der Konstruktor erhält die IP der Multicast-Gruppe als Argument. Dann muß man die Gruppennachrichten mit der Methode joinGroup() abonnieren. Dann empfängt man UDP-Nachrichten über diesen Socket wie zuvor. Bevor man die Anwendung beendet, sollte man das Abonnement mit leaveGroup() kündigen, damit der nächste Multicast-Server die Gruppennachrichten nicht länger an den eigenen Rechner weiterschickt.

```
MulticastSocket socket = new MulticastSocket(4446);
InetAddress group = InetAddress.getByName("230.0.0.1");
socket.joinGroup(group);
DatagramPacket packet
= new DatagramPacket(buf, buf.length);
socket.receive(packet);
String msg = new String(packet.getData());
socket.leaveGroup(group);
socket.close();
```

Object Serialization

Zustand des Objekts (und seiner referenzierten Teilobjekte) wird in eine Bytefolge umgewandelt.

Mit transient markierte Variablen werden nicht übertragen (z.B. Filehandle).

Nur Objekte von Klassen, die java.io.Serializable Schnittstelle implementieren (fast alle Standardklassen)

```
s = new ObjectOutputStream(new FileOutputStream(filename));
s.writeObject(new Date());
s.flush();
s = new ObjectInputStream(new FileOutputStream(filename));
Date d = (Date) s.readObject();
```

Remote Methode Invocation RMI

Aufruf von Methoden eines Objekts auf einer anderen JVM, die auf einem anderen Rechner laufen kann.

```
interface Oint extends java.rmi.Remote { void meth(); }
class O extends java.rmi.UnicastRemoteObject
     implements Oint { … }
```

Server:

```
myinst=new O();
Naming.rebind("//localhost/myinstServer",myinst);
```

Client:

```
Oint stub = (Oint)
Naming.lookup("http://myserver.com/javastuff/myinstServer");
stub.meth();
```

Wir schauen uns als Alternative später CORBA genauer an!

Miss Connectivity

In Anlehnung an Miss Liberty wurde Miss Connectivity entwickelt als ein Symbol für den freien, weltweiten Zugriff auf Daten im Internet und die Vielzahl an Möglichkeiten über das Internet zusammenzuarbeiten. Ähnlich wie sich die Informationen im Internet ständig verändern, so verändert sich auch das Erscheinungsbild von MissC. Jeder Besucher kann nicht nur MissC besichtigen, sondern er kann auch ihr Aussehen ändern. Jede Änderung

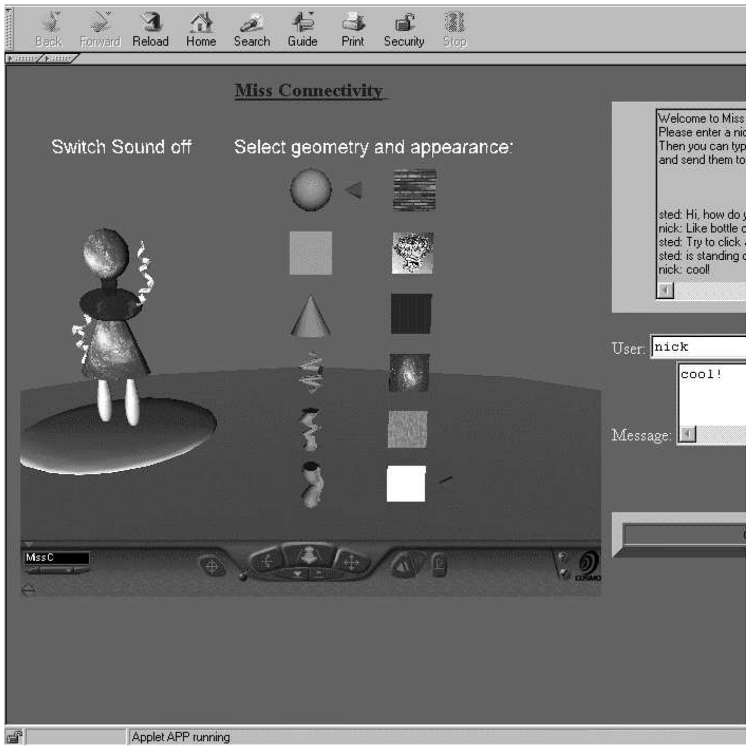
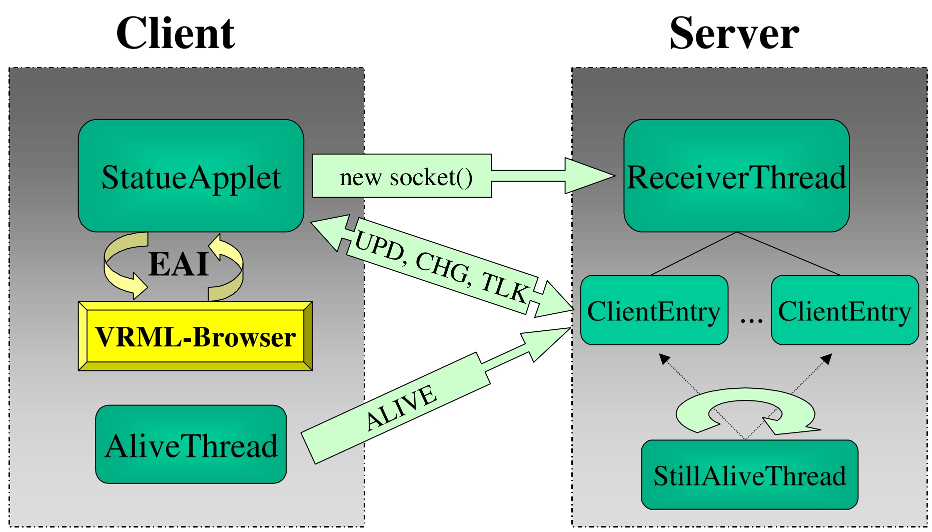


Abbildung 1: Screenshot of MissC's Webseite

wird sofort an alle anderen Besucher propagiert und die Benutzer können mittels eines Textchats ihre Änderungen an MissC koordinieren. MissC gehört zu den ersten dreidimensionalen, interaktiven, kollaborativen Kunstwerken im Internet. Sicherlich werden wir mehr ihrer Art sehen, sobald die zugrundeliegende Technologie (Netscape 4.0.x, SGI CosmoPlayer 2.x PlugIn und Java JDK1.1) reifer wird und mehr Verbreitung findet. MissC kann auf der Webseite http://www.cs.uni-sb.de/~diehl/Statue/MissC.html besichtigt werden.

In erster Linie dient MissC allerdings Ausbildungszwecken, da ihr Programmcode überschaubar ist, jedoch eine Vielzahl verschiedener Webtechnologien vereinigt: HTML, HTTP, Java-Networking, Java-Applets, Java-Threads, VRML und das EAI.



• Abbildung 2: Architektur von MissC

Die Architektur von MissC

MissC verwendet eine Client/Server-Architektur. Sie wird als HTML-Seite von einem Webserver geladen. In dieses HTML-Dokument sind eine 3D-Szene (mit <EMBED>) und ein Java-Applet (mit <APPLET>) eingebunden. In Abbildung 2 ist die Architektur und das Protokoll von MissC dargestellt. Das Java-Applet initiiert die Kommunikation mit dem Server durch Öffnen eines TCP-Sockets für den Serverport, auf dem der Server nach Anfragen lauscht. Der Server erzeugt daraufhin einen Eintrag (ClientEntry) und einen zugehörigen Thread zur Kommunikation mit dem anfragenden Client. Diesem Thread wird ein neuer Port zugeordnet und der Client und der ihm auf dem Server zugeordnete ClientEntry-Thread kommunizieren nun nur noch über diesen Port.

Zuerst schickt der Server den aktuellen Zustand (UPD) von MissC, d.h. für jeden ihrer Körperteile enthält die Nachricht das aktuelle Aussehen (Farbe,Textur) und die aktuelle Geometrie (Kugel, Box, Korkenzieher). Ändert der Benutzer ein Teil der Statue, dann schickt der Client eine Nachricht (CHG) mit der Nummer des geänderten Körperteils und des gewählten Aussehens und der gewählten Geometrie an den Server. Dieser ändert intern den Zustand der Statue und sendet die empfangene Nachricht allen anderen Clients weiter. Tippt der Benutzer eine Textzeile im Chatfenster ein, so schickt der Client eine Nachricht (TLK) an den Server, der diese an alle Clients weiterreicht. Die im Protokoll vorgesehenen Nachrichten sind wie folgt:

UPD APP1;GEO1;APP1;GEO1; …;APP2;GEO3;
CHG 1:APP1;GEO3;
TLK sted: Try to click at the plate!
ALIVE
KILL ME

Totgesagte leben länger

Da der Server lange Zeit, ideallerweise unendlich lange, laufen soll, aber dabei nicht mehr benötigte Resourcen wieder freigeben sollte, müssen Vorkehrungen getroffen werden, um festzustellen, wann ein Client die Verbindung beendet. Im einfachsten Fall teilt der Client dies explizit mit (KILL ME). Man kann aber leider nicht davon ausgehen, daß ein Client immer zum Absenden einer solchen Nachricht vor Beendigung der Kommunikation kommt, z.B. könnte ein Benutzer seinen Rechner zuvor ausschalten. Daher sendet jeder Client in regelmäßigen Abständen Lebenszeichen (ALIVE), manchmal auch als Herzschlag (heartbeat) bezeichnet. Auf dem Server läuft ein Thread der Klasse StillAliveThread, der alle Clients der Reihe nach überprüft, wann sie das letzte Lebenszeichen von sich gegeben haben. Ist dies zu lange her, wird der ClientEntry-Thread für diesen Clients beendet und der Socket (Port) wieder frei gegeben

Einige Implementierungsdetails für TCP mit Java

Wir beschränken uns aus Platzgründen hier auf den Server. Zur besseren Lesbarkeit des Quellkodes haben wir im Folgenden den Programmtext für die Ausnahmebehandlung entfernt.

Schleife des Server-Threads:

```
public void run() {
 StillAliveThread checkThread=new StillAliveThread(this);
 acceptsock=new ServerSocket(3000);
 while(true)
  { newsock = acceptsock.accept();
    clients.addElement(new ClientEntry(newsock));
    Thread.yield(); }
 }
```

Der Server erzeugt zuerst einen ServerSocket, um auf dem Port 3000 Anfragen für neue Verbindungen zu empfangen (`accept()`). Liegt eine Anfrage vor erzeugt er einen neuen Clientsrty und übergibt an diesen, den für die neue Verbindung vorgesehenen Socket `newsock`. Außerdem fügt er diesen ClientEntry der Liste der aktuellen Clients hinzu.

Konstruktor des ClientEntry-Threads:

```
public ClientEntry(Socket newsock) {
  tcpsock=newsock;
  connect();
  this.start();
}

public  void connect() {
   tcpinput = new BufferedReader(
               new
InputStreamReader(tcpsock.getInputStream()));
   tcpoutput = new
PrintWriter(tcpsock.getOutputStream(),true);
   String update="UPD ";
   update = … // aktueller Zustand
   tcpoutput.println(update);
```

```
}
```

Beim Erzeugen eines neuen ClientEntry speichert der Konstruktor den für diesen Client vorgesehenen Socket in der Objektvariablen `tcpsock`. Dann öffnet er einen Eingabe- und einen Ausgabestrom, um vom Client gesendete Daten zu lesen und Nachrichten an den Client zu schicken. Als erste Nachricht schickt der Konstruktor den aktuellen Zustand, indem er mit `println()` auf den Ausgabestrom schreibt.

Die Schleife des ClientEntry-Threads:

```
public void run() {
 boolean alive=true;
 lastAlive=System.currentTimeMillis();
 String str;
 while (alive)
 { str= tcpinput.readLine();
   if (str==null) {
     if (System.currentTimeMillis()-lastAlive)>1000)
alive=false;
     Thread.yield();
     continue; }
   if (str.equals("KILL ME")) { lastAlive=(long) 0;
alive=false;}
   if (str.equals("ALIVE"))
lastAlive=System.currentTimeMillis();
   if (str.indexOf("TLK")==0)
      StatueServer.sendUpdates(null,str);
   if (str.indexOf("CHG")==0)
        { StatueServer.sendUpdates(this,str);
          // ändere im aktuellen Zustand }

   Thread.yield(); }
}
```

Der ClientEntry-Thread liest Nachrichten vom Clients mit `readline()` aus dem Eingabestrom. Je nach Inhalt der Nachricht führt er dann verschiedene Aktionen aus. Die ersten drei IF-Anweisungen behandlen den Fall, daß die Verbindung beendet wurde. Die Nachrichten TLK und CHG werden mit der Methode `sendUpdates()` des Servers an alle anderen Clients weitergeschickt. Im Falle von TLK auch an den Clients selbst:

```
static void sendUpdates(ClientEntry sender, String str)
{ int i;
  synchronized (clients)
  { for(i=0; i<clients.size();i++)
    { ClientEntry cl=(ClientEntry) clients.elementAt(i);
      if (cl!=sender) cl.tcpoutput.println(str);
    }
  }
}
```

Da mehrere Clients gleichzeitig Nachrichten an den Server schicken können und die zugehörigen ClientEntry-Threads gleichzeitig laufen, muß man dafür sorgen, daß sie nicht konkurrierende Nachrichten an die anderen Clients schicken. Dies wird hier dadurch erreicht, daß die Sendeschleife über die Klassenvariable `clients` des Servers synchronisiert wird.

Kommunikation zwischen VRML-Browser und Java

MissC besteht im Wesentlichen aus drei Dateien: einer HTML-Datei, einer VRML-Datei und einer Java-Datei (Applet). Sie zusammen implementieren eine Webseite, in der man eine dreidimensionale Statue manipulieren und Textnachrichten eintippen kann und diese Änderungen und Nachrichten werden an alle Teilnehmer weitergeleitet, die die gleiche Seite geladen haben. MissC verwendet HTML, JavaScript, Java, VRML, das EAI und Java. Viele Technik, von der wir hier nur einen geringen Teil erklären werden, wer mehr wissen will, findet in "Java&Co" [1] zu den ersten vier Punkten und wie man sie kombiniert, die wichtigsten Informationen.

Alternativen zur Implementierung des Protokolls

Java bietet außer TCP auch andere Protokolle/Architekturen zur Netzwerkprogrammierung: URL (CGI-Skripte), UDP, Multicast UDP, RMI (Remote Methode Invocation) und CORBA (JavaIDL).

URL

Mit URL kann man eine Verbindung mit einem HTTP-Server aufbauen. Es wird eine Anfrage geschickt, der Server leitet diese an ein CGI-Skript weiter und dieses verarbeitet die Eingaben und schickt eine Antwort. Der Vorteil an URL besteht darin, daß auch Rechner hinter einer Firewall, die nur HTTP-Zugriffe erlaubt, MissC benutzen könnten. Dazu könnte man zwei CGI-Skripte schreiben: sendUpdate und getUpdate. Die Clients müssen regelmäßig das getUpdate-CGI-Skript aufrufen, um die letzten Nachrichten der anderen Clients zu erhalten, d.h. der Server schickt nicht wie bei TCP die Nachrichten automatisch an alle Clients weiter. Ähnlich wie TCP-Sockets bietet auch eine URL-Verbindung Ein- und Ausgabeströme. Im Gegensatz zur TCP-Verbindung besteht die URL-Verbindung nur für die Dauer einer Übertragung, d.h. für jede Übertragung müssen die Ströme neu geöffnet werden. Auf dem Client könnte dann folgende Methode Nachrichten an den Server schicken.

```
public void send(String upd) {
 URL script = new URL("http://www.dings.com/cgi-
bin/sendUpdate");
 URLConnection uc = script.openConnection();
 uc.setDoOutput(true);
 PrintWriter out = new PrintWriter(uc.getOutputStream());
 out.println(upd);
 out.close(); }
```

UDP und Multicasting UDP

Die Verwendung von UDP hat den Vorteil der höheren Übertragunsgeschwindigkeit und - bei Verwendung von Multicasting – der besseren Lastverteilung. Java sieht die Methoden `send()` und `receive()` zum Senden und Empfangen von UDP-Paketen vor. Um UDP einfach in die bestehende MissC-Implementierung einzubauen, könnte man eigene Ein- und Ausgabeströme implementieren, die Zeichenketten automatisch in UDP-Pakete zerlegen und umgekehrt. Verwendet man Multicasting so schickt man die Nachricht nicht mehr direkt an einen Empfänger, sondern an eine Multicastgruppe, d.h. eine spezielle IP-Adresse (224.0.0.0 bis 239.255.255.255) . Zum Empfangen von Nachrichten muß ein Empfänger zuerst Mitglied einer Multicastgruppe werden (mit der Methode `joinGroup()`) und am Ende einer Sitzung wieder aus der Gruppe austreten (mit der

Methode `leaveGroup()`). Eine entsprechende Methode eines Clients zum Empfangen von Nachrichten sieht wie folgt aus:

```
public String receive() {
  MulticastSocket socket = new MulticastSocket(4446);
  InetAddress group = InetAddress.getByName("230.0.0.1");
  socket.joinGroup(group);
  DatagramPacket packet
    = new DatagramPacket(buf, buf.length);
  socket.receive(packet);
  String msg = new String(packet.getData());
  socket.leaveGroup(group);
  socket.close();
  return msg; }
```

Verteilte Objekte (RMI oder CORBA)

Mit RMI oder CORBA ersetzt man die ASCII-Befehle des Protokolls durch Methodenaufrufe. Das Objekt, dessen Methode man aufruft, befindet sich dabei auf einem anderen Rechner und die Argumente des Methodenaufrufs werden zu diesem Rechner übertragen, die Methode dann dort ausgeführt und das Ergebnis wieder zurück an den aufrufenden Rechner übertragen. Für einfache Anwendungen verlangt RMI und vor allem CORBA zuviel Einarbeitungszeit, bietet aber auch interessante Vorteile (siehe [2]), etwa das Erweitern von Protokollen durch Vererbung und die dynamische Protokollverhandlung.

Der einfachste Ansatz besteht darin ein Anwendungsserverobjekt zu implementieren, das eine Methode `sendUpdate()` und eine Methode `getUpdate()` hat. Das Serverobjekt existiert auf dem Server und auf dem Client wird über ein Proxy-Objekt, das manchmal auch als Stub (Stumpf) bezeichnet wird, auf das Serverobjekt zugegriffen. Server- und Proxyobjekt müssen das gleiche, anwendungsspezifische Interface implementieren:

```
interface SimpleServe extends java.rmi.Remote
   { public String getUpdate();
     public void sendUpdate(String upd); }
```

Auf dem Server wird eine Klasse definiert, die das eigentliche Serverobjekt implementiert (hier stark vereinfacht). Dann wird ein solches Objekt erzeugt und bei einem Naming-Service registriert. In unserem Beispiel ist das erzeugte Objekt unter dem Namen "`SimServObj`" eingetragen.

```
class OurServer extends java.rmi.UnicastRemoteObject
      implements SimpleServe
 { String update;
   public String getUpdate() { return(update); }
   public void sendUpdate(String upd) { update=upd; }
 }

myinst=new OurServer();
Naming.rebind("//nameserver.host.com/SimServObj",myinst);
```

Der Client fragt beim Naming-Service nach, ob ein Objekt unter dem Namen `SimServObj` eingetragen ist und erzeugt dann das zugehörige Proxy-Objekt. Das Proxy-

Objekt wird explizit in den richtigen Typ umgewandet, so daß der Client die in der Schnittstelle deklarierten Methoden aufrufen kann.

```
SimleServe stub = (SimpleServe)
Naming.lookup("http://nameserver.host.com/SimServObj");

public void send(String upd) { stub.sendUpdate(upd); }
```

Implementierung verteilter, virtueller Welten mit CORBA

Bei der Implementierung verteilter Systeme haben sich Architekturen und Protokolle für verteilte Objekte und Komponenten als besonders erfolgreich erwiesen. Zu diesen gehören Sun's Java RMI, Microsoft's DCOM und der offene Standard CORBA. Im Folgenden wird CORBA näher betrachtet. Die meisten der Prinzipien findet man in ähnlicher Form in den beiden anderen Architekturen. Da es sich bei verteilten, virtuellen Welten um einen Spezialfall verteilter Systeme handelt, liegt es nahe die Dienste und das Wissen, das in eine allgemeine Architektur für verteilte Systeme geflossen ist, auszunutzen und nur dort, wo diese nicht ausreichen, die vorhandenen Dienste zu verbessern oder neue Lösungen zu entwickeln.

CORBA

CORBA wurd von der Object Management Group (OMG), die derzeit über 800 Mitglieder zählt, als Standard zur Implementierung verteilter, objektbasierter Anwendungen entworfen. Dabei wurde großen Wert auf Transparenz bestimmter Eigenschaften der verteilten Objekte gelegt. So soll die Lokalität der Objekte, d.h. auf welchem Rechner ihre Daten tatsächlich gespeichert und ihre Methoden tatsächlich ausgeführt werden, sowie die Implementationssprache der Objekte dem Benutzer eines verteilten Objekts weitestgehend verborgen bleiben, da ihn letztlich nur die Daten des Objekts und die Ergebnisse seiner Operationen interessieren. Außer einer grundlegenden Architektur für verteilte Objekte bietet CORBA auch einige standardisierte Dienste für verteilte Systeme an, zu diesen gehören eine Objektnamenverwaltung (naming), Transaktionen (transaction) und dauerhafte Speicherung von Objekten im externen Speicher (persistence).

Object Request Broker (ORB)

……..

Interface Definition Language IDL

IDL kennt die folgenden primitiven Datentypen: `short, long, float, double, char, boolean, string, octet, enum, any`. Außerdem verfügt es über eine Reihe strukturierter Datentypen, etwa `structure, union, array` (feste Länge), `sequence, exception`.

Schnittstellen (+ Vererbung)

Attribute (Lese- und Schreiboperation)

Operationen

Name, Ergebnistyp, Parameterliste (`in,out, inout`), Exceptionliste und Kontextliste

Module (Namensräume)

IDL: Ein Beispiel

```
module Beispiel{
```

```
  interface TextMsg {
    readonly attribute string msg; };

  interface EditMsg : TextMsg {

    exception notext{ string err; };

    string edit(in string newmsg)
           raises(NoTextException);
  };
};
```

IDL: Ein Beispiel

idl2java Compiler erzeugt Stubs und Skeletons:

```
_TextMsgStub.java
TextMsg.java
TextMsgHolder.java
TextMsgHelper.java
_EditMsgStub.java
EditMsg.java
EditMsgHolder.java
EditMsgHelper.java
_TextMsgImplBase.java
_EditMsgImplBase.java
```

Beispiel: Client (ORB-Initialisierung)

```
public class EditMsgClient
```

```
{ static EditMsg em_ior; // the server object

  public static void main(String args[])
  { ORB orb = ORB.init(args, null);
    org.omg.CORBA.Object objRef =
     orb.resolve_initial_references("NameService");
    NamingContext ncRef =
      NamingContextHelper.narrow(objRef);
    NameComponent nc =
      new NameComponent("EM_Obj", "");
    NameComponent path[] = {nc};
    em_ior =
      EditMsgHelper.narrow(ncRef.resolve(path));
    // Lese- und Sendeschleife !
  }
}
```

Beispiel: Client (Lese- und Sendeschleife)

```
public class EditMsgClient
{ public static void main(String args[])
  { // ORB-Initialisierung etc.
    DataInputStream data=
      new DataInputStream(System.in);
    String msg="start";
    while (!msg.equals("stop"))
     { try { em_ior.edit(msg); }
        catch (EditMsgPackage.NoTextException e )
        { System.out.println("Client: No Text!"); }
```

```
        msg=data.readLine();
      }
  }
}
```

Beispiel: Server (Object Implemenation)

```
class EditMsgServant extends _EditMsgImplBase
{ public String msg;
  public String msg() { return msg; }
  // if attribute msg were not readonly:
  // public void msg(String s) { msg=s }

  public String edit(String newmsg)
   { msg=newmsg;
     System.out.println(msg);
     return msg;
   }
}
```

Beispiel: Server (ORB-Initialisierung)

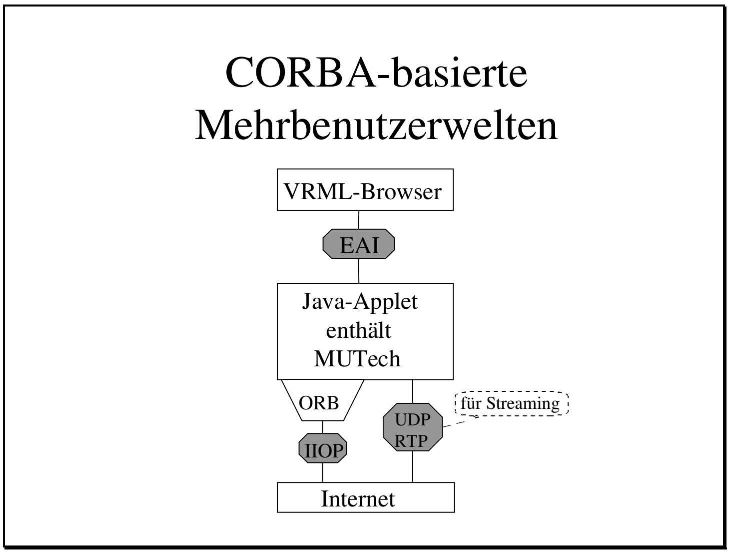
```
public class EditMsgServer {
 public static void main(String args[])
  { ORB orb = ORB.init(args, null);
    EditMsgServant em = new EditMsgServant();
    orb.connect(em);
    org.omg.CORBA.Object objRef =
      orb.resolve_initial_references("NameService");
```

```java
        NamingContext ncRef =
         NamingContextHelper.narrow(objRef);
        NameComponent nc =
         new NameComponent("EM_Obj", "");
         NameComponent path[] = {nc};
        ncRef.rebind(path, em);


         java.lang.Object sync = new java.lang.Object();
        synchronized (sync) { sync.wait(); }
       }
   }
```

Implementierung einer einfachen, verteilten, virtuellen Welt mit VRML, Java und CORBA

Wie wir gesehen haben, ist CORBA ist eine Architektur, die verteilten Objekten in heterogenen Netzen den gegenseitigen Zugriff auf deren Dienste erlaubt. Die Dienste, die ein Objekt anbietet, werden für andere Objekte (und Programmierer) in Schnittstellendefinitionen in der Sprache IDL beschrieben. Die Objekte selbst können in unterschiedlichen Sprachen geschrieben worden sein. Darüber hinaus bietet CORBA eine Vielzahl von Diensten für verteilte Systeme an.

Schaut man sich die Implementierung der MUTech im voragegangenen Kapitel genauer an, so stellt man fest, daß sie viele Verwaltungsaufgaben enthält, die in CORBA schon als Dienste vorhanden sind und teilweise effizienter implementiert sind. Denkt man darüber hinaus noch an Anforderungen, die noch zu implementieren wären (Zeit, Sicherheit, Persistenz), so stellt sich die Frage, ob es nicht geschickter wäre, CORBA zu verwenden. Einen ähnlichen, wenn auch umfangreicheren, Ansatz verfolgt das TeleVirtual Reality (TVR) Projekt an der Syracuse University und am IBM J.C. Watson Research Center.



Wie zuvor kommunizieren VRML-Browser und Applet per EAI. Für die Übertragung von zeitunkritischen Anfragen verwenden wir allerdings CORBA/IIOP. Die Kommunikation findet über einen Object Request Broker (ORB) statt. Der Programmierer schickt keine Nachrichten an andere Rechner, sondern ruft Methoden von Objekten auf, die tatsächlich auf anderen Rechnern liegen. Für Streaming von Text-, Audio- oder Video-Daten ist IIOP allerdings zu langsam, da es auf TCP aufsetzt. In diesem Fall kann man aber RTP oder ein anderes auf UDP aufsetzendes Protokoll verwenden.

Implementierung

Unser Ziel ist eine schlanke Implementierung einer MUTech, die möglichst weitgehend die Dienste von CORBA ausnutzt. So könnte z.B. unsere derzeitige Implementierung durch die Verwendung von CORBA's Event Service weiter vereinfacht werden. Dabei heißt „vereinfachen" hier, daß der Programmcode kürzer wird, er verwendet aber immer mehr und immer komplexere Mechanismen von CORBA. Da unsere MUTech als Applet implementiert ist, bedeutet jede weitere Programmzeile, daß der Benutzer länger warten muß, bis die Mehrbenutzerwelt vollständig geladen ist. Eine schlanke Implementierung bietet also auch kürzere Ladezeiten. Eine solche, schlanke Implementierung könnte auch auf verschiedene Anwendungen hin optimiert werden.

Die Sicherheitsvorkehrungen in Java-fähigen Browsern erlauben einem Applet nur, Kontakt zu dem Rechner aufzunehmen, von dem das Applet geladen wurde. Aus diesem Grund verwenden alle als Applet implementierten MUTechs einen zentralen Server. Könnte man diese Sicherheitsvorkehrungen umgehen, und dies ist im Netscape Communicator über die Klasse `netscape.security.PrivilegeManager` möglich, dann könnte man auch direkte Browser-zu-Browser Kommunikation erlauben. Für skalierbare Mehrbenutzerwelten mit Tausenden von Benutzern wäre dies ein wesentlicher Fortschritt, einzig CORBA's Naming Service dürfte zentral bleiben.

Aus oben genanntem Grund verwendet auch unsere Implementierung einen zentralen Server. Das Protokoll zwischen Client und Server legen wir anhand von Schnittstellen-Definitionen in der Sprache IDL fest. Das Protokoll kann z.B. durch Schnittstellen-Vererbung ähnlich wie in Java (interface) erweitert werden. Eine Schnittstelle besteht aus einer Menge von Signaturen. Eine Signatur wiederum besteht aus einem Methodennamen, den Argumenten und Argumenttypen, sowie dem Ergebnistyp der Methode. Mittels CORBA´s Schnittstellendatenbank (interface repository) und dem dynamischen Aufrufmechanismus (dynamic invocation) ist es prinzipiell auch möglich, daß zwei verschiedene MUTechs ihre gemeinsamen Methoden erkennen und nur über diese kooperieren.

```
module MUTech
      {   enum CommandType { changePosition, changeOrientation,
... };

             struct Command { CommandType type;
                          long id;
                          float x, y, z, angle;
                          ...  };

           struct CommandListContainer { sequence< Command >
commands; };

           interface MUTechCentral
                 {  string getInitialWorld();
                CommandListContainer getUpdates(in long id);
                void sendUpdate(in Command c, in long id);
                ...
             };
       };
```

Der Client, d.h. das Applet, kann auf den Server über einen lokalen Stub, ein CORBA-Objekt, zugreifen. Das Stub-Objekt sorgt dafür, daß die Methoden des zugehörigen Objekts auf dem Server ausgeführt werden. Dabei sind die folgenden beiden Methoden des Server-Objekts, das in den folgenden Beispielen an die Variable `centralRef` gebunden ist, wichtig:

```
centralRef.sendUpdate(change,myID);
```

Es wird die Änderung, die in `change` beschrieben wird, über den Server an die anderen Clients propagiert.

```
changelist = centralRef.getUpdates(myID);
```

Der Client erhält vom Server eine Liste aller seit der letzten Nachfrage angefallenen Änderungen.

Auf dem Server sind die Methoden des Sever-Objekts folgendermaßen implementiert.

```
public class MUTechCentralServer {
public void sendUpdate(Command c, int id)
          { int i;
             for(i=0;i<clients;i++)
          if (i!=id) ((Vector) commands.elementAt(i)).addElement(c
   );
             }

   public CommandListContainer getUpdates(int id)
          { CommandListContainer clc = new CommandListContainer();
            // kopiert alle Einträge in commands.elementAt(id) nach clc
               return clc;
          }
```

```
...

}
```

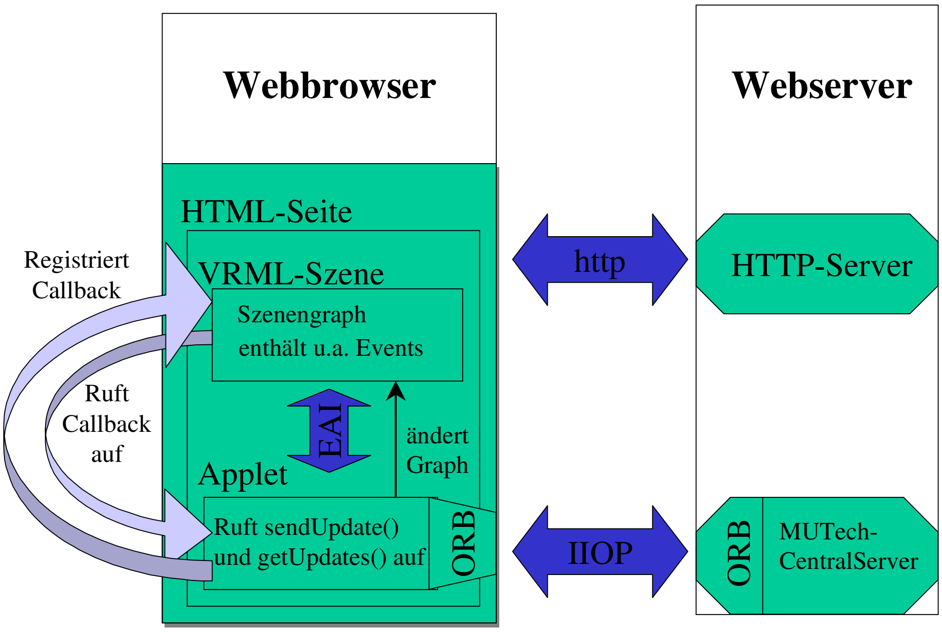
Es stellt sich nun noch die Frage, wann die obigen Methoden aufgerufen werden. Zuerst muß man über das EAI eine Callback-Methode mit jedem Event in VRML registrieren, auf das die MUTech reagieren soll. Events sind spezielle Felder von Knoten im Szenengraph. Sobald sich der Wert eines Event ändert, wird der VRML-Browser automatisch die zugehörige, registriete Callback-Methode aufrufen:

```
NODE prox = browser.getNode("PROXSENSOR");
ochange = prox.getEventOut("orientation_changed");
ochange.advise(this,null);
```

Um die Callback-Methode zu registrieren, wird ein Objekt (hier: `this`) an die `advise`-Methode des Events übergeben. Diese Objekt muß eine Methode mit der Signatur `void callback(EventOut, Object)` bereitstellen.

```
public void callback(EventOut event, double when, Object context)
{ if (etype==FieldType.SFROTATION)
   { synchronized(this)
     { ev=((EventOutSFRotation) event).getValue();
       c=new Command(CommandType.changeOrientation,
                    myID,ev[0],ev[1],ev[2],ev[3]-(float) 3.14);
       centralRef.sendUpdate(c,myID);
     }
   }
 ...
}
```

In dem obigen Programmfragment schickt die Callback-Methode den aktuellen Wert des Events `orientation_changed` an den Server mittels dessen `sendUpdate`-Methode. Wie man aus der Definition dieser Methode ersehen kann, wird der neue Wert in eine Liste (ein Objekt der Java-Klasse Vector) eingetragen. Es bleibt nun noch zu klären, wie ein Client diese Werte erhält. Da Webbrowser es aus Sicherheitsgründen Applets nicht erlauben einen Socket zu öffnen und auf diesen auf Anfragen zu warten, startet unser Applet einen Thread, der periodisch `getUpdate`-Anfragen an den Server stellt. Diese Implementierungstechnik (dumb pull) im Vergleich zu anderen (intelligent push) vergeudet zwar unnötig Bandbreite im Netz, wird aber derzeit noch durch die Browser diktiert.



Einschränkungen

Unsere derzeitige Implementierung unterstüzt kein Streaming, hier warten wir noch auf JavaSoft's Java Media API. Die Implementierung wurde durch Probleme bei der Installation der verschiedenen Softwaresysteme (z.B. Probleme mit Klassenpfaden), durch unzureichende Dokumentation und Unverträglichkeit (Java-IDL und Visigenic's ORB) derselben verzögert. Der Quellcode der MUTech umfaßt nur wenige Seiten. Derzeit überlegen wir statt Sun's Java-IDL den Internet Service Broker (ISB), der in den Netscape Communicator integriert ist, zu verwenden. Der ISB ist Visigenic's ORB in neuem Gewand. Durch die Integration in den Webbrowser reduziert sich die Menge der über's Netz zu ladenden Klassen dramatisch. Damit kann man die VRML-Dateien einer Mehrbenutzerwelt mitsamt der für Anwendung optimierten MUTech direkt vom Server laden.

Warum also CORBA?

Die wichtigsten Vorteile, die uns CORBA bietet, sind also, daß die "Signatur" des Protokolls mit IDL definiert werden kann und bei der Implementierung des Protokolls das Zerlegen von Anfragestings wegfällt. Auch könnten mit CORBA's Interface Repositories verschiedene MUTechs gemeinesame Methoden erkennen. Vor allem aber bitet CORBA Dienste, wie etwa die Namensverwaltung, für verteilte Systeme, die wir sonst selbst implementieren müßten.

Streaming

In diesem Kapitel wird die Übertragung kontinuierlicher Daten im Internet betrachtet. Es werden kontinuerliche Datenströme und ihre Eigenschaften erläutert und relevante Internetprotokolle vorgestellt. Zum Abschluß wird ein Mechanismus zur kombinierten Fehler- und Jitterkontrolle beschrieben.

Kontinuierliche Datenströme

Die klassischen Vertretern kontiniuerlicher Datenströme gehören Audio- und Videoübertragungen. Kontinuierliche Daten fallen aber auch in anderen Anwendungen an, so z.B. bei der Übertragung von Meßdaten oder Kontrolldaten (Fernsteuerung). Kontinuierliche Datenströme haben meist folgende Eigenschaften. Sie unterliegen strengen Zeitanforderungen, d.h. Daten, die zu spät ankommen, sind nutzlos. Sie sind jedoch meist tolerant gegenüber geringen Datenverlusten, dies hängt jedoch stark vom Medium, der Kodierung der Daten und der menschlicher Wahrnehmung ab. Schließlich weisen die Daten meist eine Periodizität auf, d.h. z.B. daß eine feste Anzahl Bilder pro Sekunde (samples oder frames per second) gesendet wird.

In der Regel werden die Daten komprimiert über das Netz geschickt und zur Komprimierung ein für das Medium spezifisches Verfahren verwendet. Grob kann man diese Verfahren in solche unterteilen, die Samples (zu einem Zeitpunkt erfaßte Daten) unabhängig von anderen Samples kodieren, z.B. Motion-JPEG, und solchen, die Differenzen zwischen Samples übertragen, z.B. MPEG. Das letztere Verfahren ist in der Regel anfälliger gegenüber Verlust von Daten. Orthogonal dazu gibt es Verfahren, die eine hierarchische (layered, scalable, sub-band coding) Kodierung verwenden. Dabei werden zu einem Sample Daten unterschiedlicher Güte geschickt: Auf der untersten Ebene wird ein Leitsignal geschickt, es genügt zur Wiedergabe mit geringer Qualität, in anderen Ebenen werden Differenzen hierzu oder zusätzliche Daten geschickt, diese ermöglichen eine verbessert Qualität (z.B. höhere Abtastrate oder Auflösung) bei der Wiedergabe.

Untersuchungen existierender Systeme für Audio/Video-Konferenzen über das Internet haben die drei wesentlichen Ursachen für die meist schlechte Qualität ausgemacht. Zum einen tritt bei Verwendung unzuverlässiger Protokolle ein Verlust von Datenpaketen auf oder bei Verwendung zuverlässiger Protokolle werden die Übertragungszeiten zu hoch. Ein wesentlicher Faktor ist aber auch der Einsatz von Multitasking auf PCs und Workstations. Die dort installierten Betriebssysteme bieten bisher noch kein Multitasking, das Echtzeitanforderungen gerecht wird. Schließlich wurden noch eine Vielzahl akustischer und optischer Probleme festgestellt, die teilweise durch schlechte Hardware verursacht wurden.

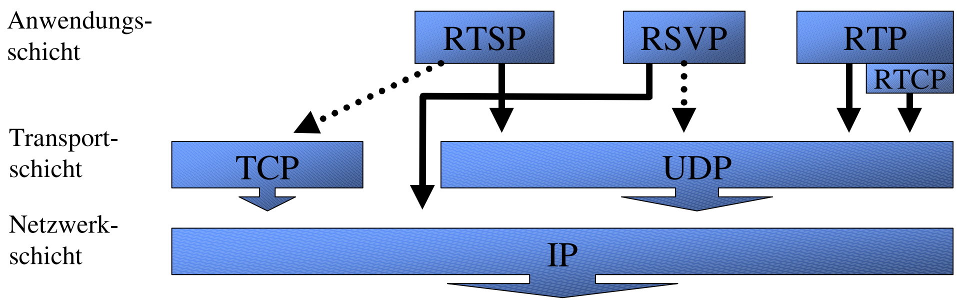
Kontinuierliche Datenströme in verteilten, virtuellen Welten

In verteilten, virtuellen Welten kann man die Positionsänderungen der Avatare als kontinuierliche Datenströme auffassen, allerdings bieten sich meist andere Techniken an, etwa dead reckoning wie in DIS, oder das Samplen wie in VSPLUS. Besonders viele Bewegungsänderungen fallen etwa an, wenn der Avatar durch unkonventionelle Eingabegeräte, wie etwa bei Peloton, gesteuert wird. Kontinuierliche Datenströme spielen aber aber auch eine zentrale Rolle, wenn man live Videoaufnahmen der Gesichter der Benutzer auf die Köpfe ihrer Avatare projizieren will oder wenn man Audiochat in einer Welt anbieten will. Einige VRML-Browser, wie etwa SGI's CosmoPlayer, unterstützen bereits 3D-Sound, d.h. man kann mehrere Klangquellen mischen und dabei wird die Entfernung und die Ausrichtung dieser Klangquellen sowie die aktuelle Position des Betrachters berücksichtigt.

Relevante Internetprotokolle

	Zuverlässiges Multicast	Echtzeit-Multicast
Eine Quelle 1 zu N	Multicast-FTP Software-Updates	A/V-Konferenzen Internetradio, -kino
Mehrere Quellen N zu M	CSCW Verteiltes Rechnen	DIS Verteilte, virtuelle Welten

Bei der Übertragung von kontinuierlichen Datenströmen in Audio/Videokonferenzen, aber auch in verteilten, virtuellen Welten hat sich Multicasting als Mittel der Wahl zum Einsparen von Übertragunsbandbreite bewährt. Verteilte, virtuelle Welten fallen in die Klasse der Anwendungen, bei denen mehrere Datenquellen, in die Browser aller Benutzer, vorliegen und an alle Benutzer geschickt werden. Dabei hat die Übertragungsgeschwindigkeit (Echtzeit) Vorrang vor der Zuverlässigkeit der Übertragung. Im Folgenden betrachten wir einige Internetprotokolle, die für solche Anwendungen eine Rolle spielen.



Im Einzelnen betrachten wir die Protokolle RSVP (Resource Reservation Protocol), RTSP (Real-Time Streaming Protocol), RTP (Real-Time Transport Protocol) einschließlich RTCP (RTP Control Protocol).

RSVP: Resource Reservation Protocol

Das RSVP bietet ein erweitertes Multicasting. Jeder MRouter hat zusätzliche Datenstrukturen (priority queues). Der Empfänger schickt Reservierungsanfrage zum Sender. An jedem MRouter-Knoten auf dem Weg zum Sender wird die angeforderte Bandbreite reserviert. Falls ein MRouter nicht mehr über genügend Bandbreite verfügt, schickt er eine Fehlermeldung an den Empfänger. Mit anderen Worten, sobald die Reservierungsanfrage beim Sender ankommt, wurde ein virtuelle Verbindung für alle weiteren Multicast-Pakete zwischen dem Empfänger und dem Sender durchgeschaltet und die erforderliche Bandbreite reserviert.

RTSP: Real-Time Streaming Protocol

Das RTSP dient zur Kontrolle mehrerer Sitzungen, bei denen Medienströme, in der Regel also Audio und Video, übertragen werden. Die Funktionalität von RTSP kann mit der einer Fernbedienung verglichen werden, es wird festgelegt, was gesendet werden soll und die Übertragung kann gestartet, unterbrochen, fortgesetzt oder abgebrochen werden. In Spezifikation von RTSP werden die Anfragen und Antworten als Methoden bezeichnet. Unter anderem sind folgende Methoden in RTSP vorgesehen:

- DESCRIBE: Die Beschreibung eines Mediums wird vom Server geladen.

- ANNOUNCE: Ein Client schickt hiermit die Beschreibung eines Mediums an einen Server, der diese in die Liste, der ihm bekannten Medien eintragen kann. Außerdem kann ein Server hiermit den Client benachrichtigen, daß sich eine zuvor mit DESCRIBE abgefragten Beschreibung geändert hat.

- SETUP: Der Client schickt eine Liste akzeptierbarer Transportmechanismen an den Server. Die Antwort des Servers enthält den ausgewählten Mechanismus. Nun kann der Client mit PLAY die Übertragung des Medienstroms starten. Man kann also mit SETUP den Übertragungskanals (UDP, multicast UDP, TCP) und den Übertragungsmechanismus (z.B. RTP-basiert) auswählen.

- PLAY: Der Client schickt eine solche Anfrage an den Server, damit dieser mit der Übertragung der Daten beginnt.

- PAUSE: Zum Unterbrechen der Übertragung wird diese Methode an den Server geschickt. Die Übertragung kann später durch die Methode PLAY an der Position fortgesetzt werden, an der sie zuvor unterbrochen wurde.

- TEARDOWN: Der Client fordert hiermit den Server auf, die Übertragung zu beenden. Der Server kann daraufhin alle Resourcen frei geben.

- REDIRECT: Der Server teilt dem Client mit, daß er für dieses Medium einen anderen Server kontaktieren muß.

RTP: Real-Time Transport Protocol

RTP wurde zur Übertragung von Daten mit Echtzeitanforderungen an mehrere Teilnehmer entworfen. Anstatt mehrere Medien, z.B. Video und Audio, auf dem Server zu verschmelzen und dann als ein Strom über das Netz zu übertragen, wurde bei RTP bevorzugt, daß für jedes Medium ein separater Datenstrom geschickt wird. Dies erlaubt es dem Empfänger nur einige der Datenströme auszuwählen, allerdings müssen die Datenströme beim Empfänger synchronisiert werden. RTP dient also dem Verpacken und Transportieren der Daten. Zusätzlich wurde in der Spezifikation von RTP (RFC 1889) das RTCP (RTP Control Protocol) definiert. Es erlaubt die Überwachung (Monitoring) der Übertragungsqualität (quality of service QoS) und bietet eine primitive Sitzungskontrolle, d.h. es führt Buch,wer einer Sitzung beitritt und wer sie verläßt.

RTP-Pakete werden meist als UDP-Datagramme geschickt. Ein RTP-Paket besteht aus einem Kopf und einem Rumpf, der sogenannten payload. Der Rumpf enthält die eigentlichen Daten, also z.B. Audio- oder Videoframes. Im Kopf wird der Typ der Daten (7 Bits), der sogenannte payload type, eine Sequenznummer (16 Bits), ein Zeitstempel (32 Bits) und die ID des Mediums (32 Bits), die sogenannte sychronization source ID, kurz SSRC, angegeben. Für zusammengemischte Medien kann noch eine Liste der IDs, der Teilströme folgen, die sogenannte contributing sources list, kurz CSRC list. RTP ist erweiterbar, indem man neue payload types definieren kann. Für jeden payload type muß man dann auch das Format für die zugehörige payload, also den Rumpf des Pakets, festlegen.

RTCP

Wie bereits erwähnt, dient das RTCP dem Überwachen (monitoring) der Übertragungsqualität. RTCP kennt fünf verschiedene Pakettypen: Senderberichte, Empfängerberichte, Quellenbeschreibungen, Sitzungsende und anwendungsspezifische Informationen. Wir schauen uns hier einmal den Inhalt eines Empfängerberichts genauer an. Empfängerberichte enthalten die SSRC des empfangenen Medienstroms, den prozentualen und den gesamten Verlust an Datenpaketen. Die höchste empfangene Sequenznummer, eine Abschätzung für die schwankende Empfangsverzögerung (“interarrival jitter”, wird später noch genauer erklärt), sowie den Zeitstempel des zuletzt empfangenen Senderberichts. Aus den Informationen der Sender- und Empfängerberichte lassen sich sowohl beim Sender als auch beim Empfänger wichtige Qualitätsinformationen berechnen, etwa die Transportverzögerung, d.h. die Übertragunszeit für ein Paket, die Paketverlustrate oder die mittlere payload Datenrate, d.h. wieviele Bits der eigentlichen Mediendaten werden pro Sekunde übertragen. Aufgrund dieser Berechnungen kann der Sender den Übertragungsmode ändern, oder der Empfänger kann Probleme erkennen und einen anderen Sender selektieren. Anstatt die Informationen sofort auszunutzen, können auch über längere Zeit Statistiken erstellt werden, die den Administratoren helfen können, das Netz zu evaluieren.

Spezielle RTP-Programme

In der Regel kommunizieren Sender und Empfänger über zwei Multicast-Adressen miteinander. Die erste Multicast-Adresse dient der Übertragung der RTP-Pakete, die zweite der Übertragung der RTCP-Berichte. Aufr einem der Rechner zwischen Sender und Empfänger können spezielle RTP-Programme laufen. Die RTP-Spezifikation unterscheidet klassifiziert diese Programme als Mixer, Translator und Monitor. Ein Mixer mischt Ströme, die RTP-Pakete, die der Mixer weiterschickt, erhalten die SSRC des Mixers und CSRC Liste enthält die SSRC der Quellenströme. Ein Mixer könnte z.B. mehrere Audioströme zur Übertragung über eine Verbindung mit niedrigerer Bandbreite zusammenmischen. Ein Translator kann die Kodierung der payload oder einer Protokollebene ändern, etwa um die Pakete durch eine Firewall zu schleusen. Im Gegensatz zum Mixer behalten die Pakete die ursprüngliche SSRC. Ein typischer Translator ist ein Replikator, dieser kopiert die Pakete und schickt sie über Unicast an mehrere Empfänger weiter. Ein Monitor empfängt die RTCP-Pakete und schätzt die Übertragungsqualität ab, füht eine Fehlerdiagnose durch oder erstellt Statistiken für Administratoren. Monitore können als eigenständige Anwendungen implementiert sein, sie können aber auch Teil der übertragenden Anwendungen sein und diese in die Lage versetzen, Übertragungsmodalitäten anzupassen.

Fehlerbehandlung bei Datenströmen

Bei der Übertragung von Datenpaketen können diese verloren gehen. Eine Anwendung kann einen solchen Datenfehler grundsätzlich auf drei Arten behandeln. Sie kann ihn ignorieren, sie kann ihn vertuschen, oder sie kann ihn korrigieren. Zur Fehlerkorrektur (error recovery) gibt es zwei Ansätze, entweder man sendet genügend redundante Informationen, so daß aus diesen die verlorene rekonstruiert werden kann, oder die verlorengegangenen Daten werden erneut übertragen. Um Fehler zu vertuschen (error concealment), kann man z.B. bei Videobildern Interpolation verwenden. Verfahren zur Fehlervertuschung hängen stark von der Anwendung ab.

Forward Error Control

Das folgende Beispiel zeigt, wie man durch ein drittes Paritätsbit, Datenpakete, die aus zwei Bits bestehen, so erweitern kann, daß, wenn eines der drei Bits verloren geht, man aus den verbleibenden beiden Bits, das verlorene rekonstruieren kann.

Erstes Bit	**Zweites Bit**	**Paritätsbit**
0	0	0
0	1	1
1	0	1
1	1	0

Das Verfahren der Paritätsbits läßt sich erweitern. FEC-Verfahren berechnen zu k Datenpakete und h Paritätspakete. Man bezeichnet das Verhältnis h/k als overcode (Kodierungsoverhead). Wenn nicht mehr als h Pakete der $h+k$ Pakete verloren gehen, kann der Empfänger die k Pakete rekonstruieren.

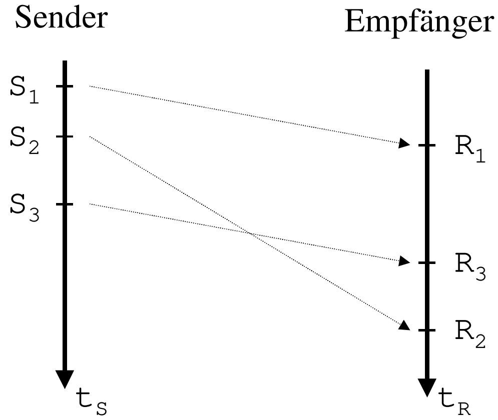
Hybride Verfahren kombinieren FEC mit erneutem Senden von Daten, z.B.:

- Es werden die k Datenpakete geschickt. Nur bei Datenverlust werden anschließend die h Paritätspakete geschickt.

- Es werden die `k Datenpakete und die h` Paritätspakete geschickt. Falls davon mehr als h Pakete verloren gehen, werden die verlorengegangenen Pakete erneut geschickt.

Interarrival Jitter

Mit Jitter (Englisch für Schwankungen) bezeichnet man die mittlere Abweichung der Zeitspannen zwischen den Absendezeitpunkten und den zugehörigen Empfangszeitpunkten zweier Pakete. Sei S_i der Sendezeitpunkt des i-ten Pakets und R_i dessen Empfangszeitpunkt.



Dann läßt sich die Zeitspanne zwischen dem Senden und dem Empfangen zweier Pakete wie folgt berechnen:

$$D(i,j) = (R_j - R_i) - (S_j - S_i)$$
$$D(i,j) = (R_j - S_j) - (R_i - S_i)$$

Beide Formeln sind äquivalent. Die erste berechnet, die Differenz wie in der obigen Definition von Jitter. Die zweite hingegen berechnet die Differenz der Übertragungszeiten für beider Pakete. Der erste Ansatz ist für uns hier interessant, da man die Zeitstempel von Paketen hierzu verwenden kann, bzw. die Zeitpunkte, zu denen die Pakete ankommen. Da die Uhren auf dem Sender und dem Empfänger nicht absolut synchron laufen, kann man R_j-R_i bezüglich der Zeit des Empfänger berechnen und erhält eine Zeitdauer und S_j-S_i bezüglich der Zeit des Senders und erhält eine Zeitdauer. Nach der Übertragung von n Paketen erhält man dann für den Jitter:

$$J = \frac{\sum_{i=2}^{n} |D(i-1,i)|}{n}$$

Da man den Jitter während der Übertragung als Maß für die Übertragungsqualität verwenden will, aber ja noch nicht alle zukünftigen Pakete vorliegen, berechnet man in RTP eine Abschätzung zur Übertragungszeit:

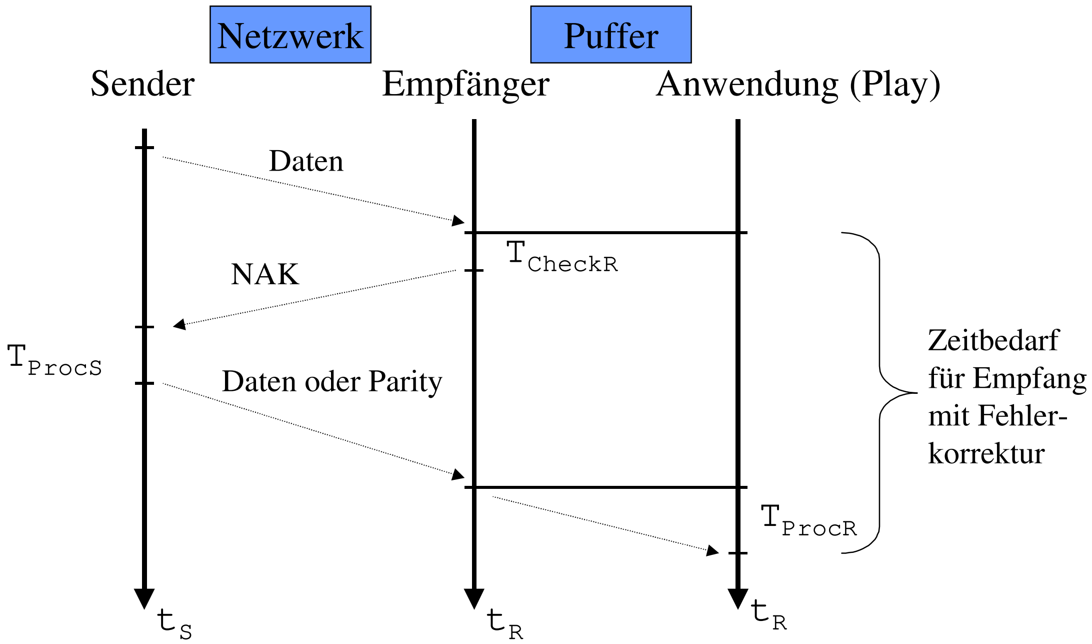
$$\tilde{J}(i) = \tilde{J}(i-1) + \frac{|D(i-1,i)| - \tilde{J}(i-1)}{16}$$

Dabei geht die aktuelle Differenz nur zu einem Sechszehntel in die Abschätzung ein, damit eine kurzfristige, starke Schwankung die Abschätzung nicht zu stark verändert (Rauschunterdrückung).

Kombinierte Jitter- und Fehlerkontrolle

Um Paketverluste auszugleichen, verwenden Anwendungen wie etwa Audiochat oder Videostreaming im Internet Empfangspuffer, in denen erst eine Reihe von Paketen empfangen, richtig sortiert und gegebenenfalls verlorengegangene Pakete erneut angefordert werden können. Es stellt sich die Frage, wie groß so ein Puffer sein muß. Im Folgenden betrachten wir einen adaptiven Empfangspuffer, der seine Größe anhand der aktuellen Übertragungszeiten und der obigen Jitter-Abschätzung anpaßt.

$$T_{Buffer} = T_{CheckR} + T_{NAK} + T_{ProcS} + T_{Parity} + T_{ProcR} + \frac{3 * \tilde{J}}{2}$$



Auf Sender und Empfänger haben wir zwei unterschiedliche Zeitachsen t_S und t_R, für die Berechnungen betrachten wir immer Zeitintervalle, die jeweils bzgl. einer dieser Zeitachsen ausgerechnet werden. Wir betrachten den Fall, daß ein Paket verloren geht oder fehlerhaft ankommt. Um dies festzustellen, benötigt der Empfänger die Zeit T_{CheckR}, daraufhin schickt er eine Nachricht (NAK, negative acknowledgement) an den Sender, damit dieser das Paket erneut schickt. Dieser braucht die Zeit T_{ProcS}, um das Paket die Anfrage zu verarbeiten, dann schickt er das Paket oder ein Paritätspaket erneut – möglicherweise sogar zuverlässig über TCP. Der Empfänger benötigt nun die Zeit T_{ProcR}, um das Paket oder das Paritätspaket zu verarbeiten. Beachtet man zusätzlich noch den Jitter, der dreimal eine Rolle spielt: beim Empfang der ursprünglichen Daten, beim Senden des NAK und beim erneuten Empfang des Pakets, so erhält man folgende Abschätzung für den Zeitbedarf für des Empfangen eines Pakets mit einmaliger Fehlerkorrektur:

Damit sollte die Anzahl von Paketen im Puffer mindestens k betragen, wobei T_{Sample} die Zeitspanne ist, die zum Abspielen eines Pakets benötigt wird.

$$k = \frac{T_{Buffer}}{T_{Sample}}$$

Die Zeitspannen T_{CheckR}, T_{ProcS} und T_{ProcR} sind fix, alle anderen können anhand der Zeitstempel geschätzt werden.

Partitionierung verteilter, virtueller Welten

Wie Informatiker die Welt unter sich aufteilen …

Inhalt:
In diesem Kapitel werden Techniken zur räumlichen Aufteilung von Szenen vorgestellt. Zu diesen gehören LODs, Bounding Boxes, Binary Space Partitioning Trees und Cells und Portals. Anschließend werden dann Verfahren vorgestellt, die speziell zur räumlichen Aufteilung für Mehrbenutzerwelten entwickelt wurden: Filtering, "Locales and Beacons" und Auren.

Räumliche Aufteilung
Normalfall: VRML-Browser muß ganze Szene laden und rendern

Spational Partitioning (räumliche Aufteilung) = teile 3D-Raum in Teilräume und berechne oder definiere Sichtbarkeitsrelation (Raum-zu-Raum)

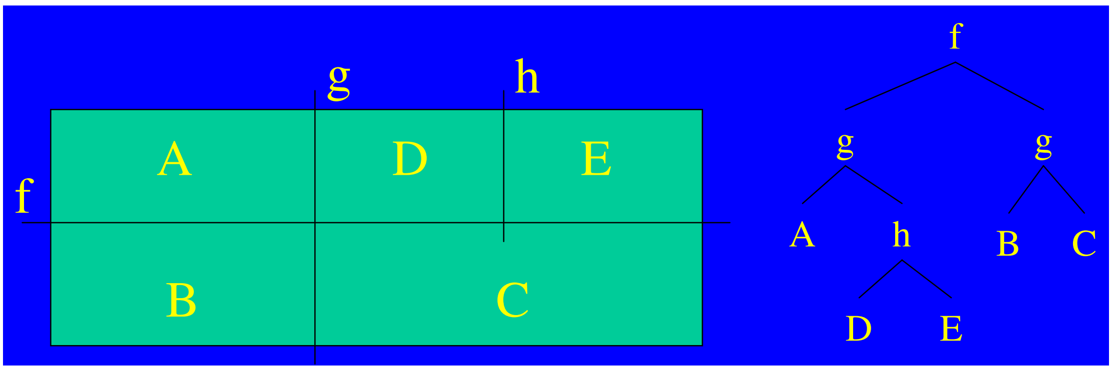
? nur sichtbare Teilräume müssen geladen und gerendert werden

LOD
In Abhängigkeit von der Entfernung des Betrachters vom Objekt, wird der Szenengraph für das Objekt ausgewählt. Je näher, um so genauer.
Zur Auswahl der Detailgenauigkeit, bzw. Sichtbarkeit innerhalb von Gebäuden unbrauchbar.

```
LOD { range [ 10 100 1000 ]
   level [ USE O1  USE O2
       USE O3  USE O4 ]}
```

Bounding Boxes
Teilraum ist in einem SWITCH-Knoten und wird durch einen ProximitySensor an- bzw. ausgeschaltet.
Die Bounding Box des ProximitySensors und des Teilraums stimmen überein, d.h. erst wenn, der Betrachter sich in diesen Teilraum hineinbewegt, wird dieser gerendert.
Eine Staffelung (ähnlich LOD) von ProximitySensoren und Bounding Boxes ist denkbar.
BBs auch für Culling verwendbar!
Konservativ: Abschätzung der sichbaren Polygone

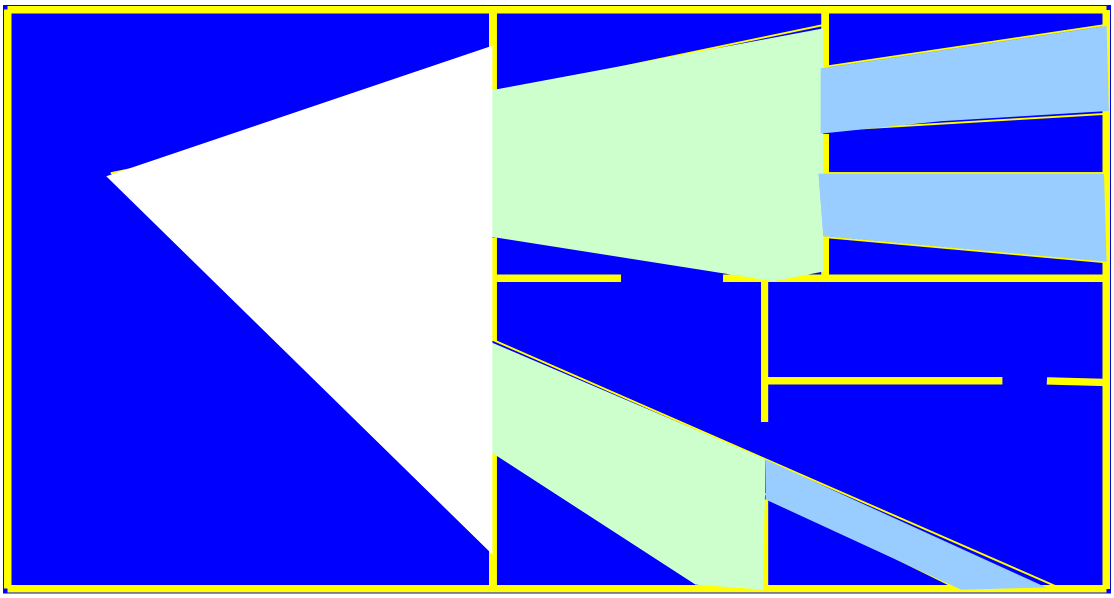
BSP Trees
BSP = Binary Space Partitioning
Teilt Raum hierarchisch durch Ebenen in Teilräume auf
Hier: 2D Beispiel (Geraden statt Ebenen)



BSP Trees
Zwischenknoten: Ebenen, gegeben durch Parameter a,b,c,d für Ebenengleichung
a*x+b*y+c*z+d=0

linker Teilbaum für alle Punkte (x,y,z) , für die a*x+b*y+c*z+d<0
linker Teilbaum für alle Punkte (x,y,z) , für die a*x+b*y+c*z+d>0
Blattknoten: Teilräume
Suche in BSP Tree:
geg.: Punkt (x,y,z), ges.: Teilraum, in dem dieser Punkt liegt
BSP Trees und Sichtbarkeit
Die folgende Technik wurde so z.B. im Computerspiel Quake verwendet.
Blattknoten: Teilraum + Liste aller Teilräume, die von diesem aus sichtbar sind. (statisch, von Hand eingetragen!)
Technik:Berechne mit Hilfe des BSP Trees, in welchem Teilraum sich der Betrachter befindet, rendere dann diesen Teilraum und alle von diesem aus sichtbaren Teilräume.
Implementierung in VRML: Betrachterposition durch ProximitySensor ermittelt an Script routen, in dem BSP Tree durchsucht wird, dann zu rendernde Teilräume über SWITCH-Knoten anschalten.

Cells and Portals
Konservative Berechnung der Sichtbarkeit
Anwendung: Gebäude, Fahrzeuge, Höhlen, etc.:
bestehen aus Teilräumen (Cells=konvexe Polyhedrons) und sind durch Fenster, Türen oder sonstige Öffnungen (Portals=Polygone) miteinander verbunden.
Eine Zelle ist von einer anderen nur durch eine Folge von Portals sichtbar.
Zwei Lösungen:
Statische Raumaufteilung: langsame, exakte Vorberechnung (Jones, Teller)
Dynamische Raumaufteilung: schnelle, approximative Sichtbarkeitsberechnung (Luebke&Georges)



Cells and Portals: Basisalgorithmus

Portal: `(p,c,d)` mit `p` Polygon des Portals, verbindet Zelle `c`
mit Zelle `d`.
`VisCells(c,V)` berechnet alle von `c` im Blickfeld `V` sichtbaren Zellen.

`Restrict(V,p)` berechnet Einschränkung des Blickfelds `V`
durch das Polygon `p`
`VisPortals(c,V)` berechnet alle Portals, die in Zelle `c`
im Blickfeld `V` liegen.

Sei `c` Zelle, in der der Betrachter ist und `V` sein Blickfeld.
Berechne `VisibleCells(c,V)`.

$$\mathrm{VisCells}(c,V) = \{c\} \cup \bigcup_{(p,c,d)\in \mathrm{VisPortals}(c,V)} \mathrm{VisCells}(d, \mathrm{Restrict}(V,p))$$

Räumliche Aufteilung für Mehrbenutzerwelten
Skalierbarkeit
Reduzieren der erforderlichen Bandbreite
Verteilen der Teilräume auf unterschiedliche Server
Reduzieren des Renderingaufwands

Einsicht: Ein Benutzer ist in der Regel nur an dem interessiert, was in seiner nächsten Umgebung passiert.

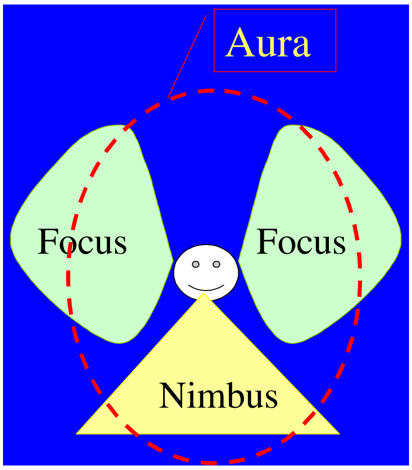
Filtern (z.B. in DIS)
Ein Rechner dient als Filter und leitet nur bestimmte Nachrichten an seine Clienten weiter.
Parameter
Schärfe (acuity) = Größe/Entfernung vom Betrachter
Nur Objekte mit für die dieses Verhältnis größer ist, sind sichtbar und ihr Szenengraph muß an den Client geschickt werden.
Max. Anzahl der Objekte, für die der Client Änderungen empfangen kann/will.
Teilraum, in dem der Client ist. Nur Änderungen in Teilräumen, die von diesem aus sichtbar sind, müssen an den Client geschickt werden.

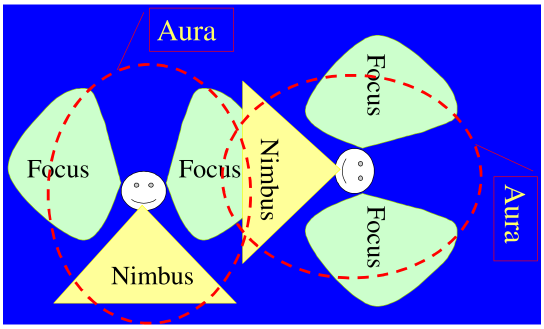
Locales and Beacons
Verwendet in SPLINE (Scalable Platform for Large Interactive Networked Environments) von MERL (Mitsubishi Electric Research Laboratory) --> Diamond Park
Locale (Schauplatz)
eigene Menge von Multicast-Adressen
eigenes Koordinatensystem
Benutzer sieht das Locale, in dem er sich befindet und die benachbarten Locales. (können überlappen!)
Beacon (Leuchtfeuer)
Namensserver liefert Locale, in dem das Objekt mit diesem Namen (tag) sich gerade befindet.

Locale
Instanzvariablen einer Locale L
eindeutige ID für L
Vaterobjekt
Verschachtelung von Locales möglich
z.B. Auto hat eigenes Locale und bewegt sich in einem anderen
3D Volumen von L
Multicast-Adressen von L
Für jedes benachbarte Locale N:
ID von N

Transformationsmatrix: N nach L
Transformationsmatrix: L nach N

Aura
Verwendet in den Systemen MASSIVE und DIVE
Jedes Objekt hat pro Ereignisklasse (Medium) folgende Eigenschaften:
Aura = 3D-Volumen, in dem Objekt mit anderen Objekten interagieren kann, d.h. wenn sich die Auras zweier Objekte schneiden, können die beiden Objekte miteinander interagieren.
Focus = 3D-Volumen, in dem das Objekt Ereignisse (akustisch, visuell, …) wahrnimmt.
Nimbus = 3D-Volumen, in dem von dem Objekt verursachte Ereignisse wahrnehmbar sind.





Gegenseitige Wahrnehmung

Objekt A		Objekt B	
Focus	Nimbus	Focus	Nimbus
-	-	-	-
B	-	-	-
-	B	-	-
B	B	-	-
B	-	A	-
-	B	A	-
B	B	A	-
-	B	-	A
B	B	-	A
B	B	A	A

10 nicht symmetrische Fälle

Für jeden Fall (Wahrnehmungsgrad) kann man unterschiedliche Darstellung des Objekts oder gar verschiedene Räume verwenden.
Funktionsweise:
Sobald ein Objekt A mit dem Focus eines Objekts B kollidiert, erhöht sich die A's Wahrnehmung von B.
Kollidiert A mit dem Nimbus von B, so erhöht dies B's Wahrnehmung von A.
Beispiel: A nimmt B wahr --> A lächelt

Aura-Manager
AMs zuständig für Bereiche (Quadranten) der verteilten Szene
Informiert Objekt A, wenn ein anderes Objekt B mit seiner Aura kollidiert. Objekt A kann nun B kontaktieren und z.B. dessen Szenengraph laden.
hierarchisch: Master AM übergibt Objekt an anderen AM, sobald es einen Quadranten verläßt und in einen anderen eintritt.

[redacted]