Johann Wolfgang Goethe-Universität

Frankfurt am Main

Fachbereich Biologie und Informatik

Institut für Informatik

Professur für Graphische Datenverarbeitung

Diplomarbeit

Entwurf und Implementierung eines Informationsvisualisierungssystems auf Basis der Relevanzkugel-Metapher unter Verwendung von VRML und Java

Klaus Mittag

Matr.-Nr. 843746

März 2005

Eingereicht bei Prof. Dr.-Ing. Detlef Krömker

Betreuer: Prof. Dr.-Ing. Detlef Krömker

Externe Betreuer: Prof. Dr.-Ing. Matthias Hemmje, Gerald Jäschke

Kurzfassung

Visualisierungssysteme nutzen die Mittel der modernen Computergraphik, um Informationen und Zusammenhänge zu veranschaulichen. Ein wichtiges Teilgebiet besteht dabei in der Veranschaulichung großer Informationsmengen zur Gewinnung eines Überblicks und Vorauswahl potentiell interessanter Teilmengen, die dann mit weiterführenden Methoden im Detail erforscht werden können.

Das Relevanzkugelmodell wurde erstmals eingeführt, um als Bestandteil des LyberWorld-Projekts genau diese Vorselektion auf einer Menge von Textdokumenten zu leisten. Ziel dieser Arbeit ist es, dieses Modell in eine neue Form auf Basis des World Wide Web zu überführen und damit aus der engen Anbindung an das ursprüngliche System zu lösen und allgemeiner verwendbar zu machen. Zu diesem Zweck werden zunächst das Modell an sich und seine früheren Implementierungen genauer betrachtet, dann nach Auswahl geeigneter Hilfsmittel – VRML zur graphischen Modellierung und Java zur Handhabung der Funktionalität – Konzepte zur weiteren Ausgestaltung und zur Behebung existierender Schwächen des Ansatzes erarbeitet, und schließlich die resultierende Implementierung beschrieben und bewertet.

Erklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig verfaßt und keine Quellen oder Hilfsmittel als die in dieser Arbeit angegebenen verwendet habe.

Frankfurt am Main, den 02. März 2005

Klaus Mittag

Inhaltsverzeichnis

Einleitung

Motivation

Eines der zentralen Probleme der heutigen Informationsgesellschaft besteht darin, aus dem vorherrschenden Überangebot an prinzipiell verfügbaren Informationen diejenigen herauszufiltern, die für den Einzelnen tatsächlich von Interesse sind. Beispiele dafür finden sich im weitgefächerten Spektrum der klassischen Rundfunk- und Druckmedien ebenso wie bei der Benutzung von großen Datenbanken und der Informationssuche im Internet. Es liegt in der Natur solcher großen Informationssammlungen, daß sich ihr Angebot durch das Hinzukommen neuer Daten, die Weiterverarbeitung bereits existierender Bestände, und das Herausfallen veralteter und überholter Informationen praktisch ständig im Umbruch befindet. Damit wird aber die *Suche* nach aktuellen Informationen leicht zu einer echten Herausforderung für den Suchenden, während auf der anderen Seite der einzelne Informations*anbieter* mit dem Problem konfrontiert ist, sein beabsichtigtes Publikum möglichst effektiv zu erreichen und ihm den Zugriff so einfach wie möglich zu machen.

Speziell im Bereich der Informationstechnologie gibt es zu diesem Zweck eine Fülle von verschiedenen Ansätzen und Methoden, Daten zu speichern, wiederzufinden, zu verwalten, und bei Bedarf wieder zugänglich zu machen. Was als reines Forschungsgebiet für Computerspezialisten begann, wird heute im Rahmen der weiten Verbreitung von elektronischer Datenverarbeitung und der zunehmenden weltweiten Vernetzung solcher Systeme untereinander mehr und mehr auch eine Aufgabe des täglichen Lebens. Der duchschnittliche Anwender auf der Suche nach elektronisch gespeicherten Informationen (oder auch nur nach *Querverweisen* auf Daten, die dann in anderer Form vorliegen) wird aber normalerweise kaum die Zeit und/oder Neigung haben, sich erst umständlich in ein kompliziertes System mit einer eigens zu erlernenden Sprache einzuarbeiten, um dann endlich mit seiner *eigentlichen* Arbeit fortfahren zu können[1]. Daraus motiviert sich die Suche nach Werkzeugen, die dem menschlichen Anwender die Informationssuche idealerweise so weit erleichtern, daß er sie nicht mehr in erster Linie als ärgerliche Störung bei der Verfolgung seiner Absichten empfindet. Von den zu diesem Zweck entwickelten

[1] Zum Beispiel ist die Formulierung einer etwas komplexeren SELECT-Suchanfrage in der Standardsprache SQL schon beinahe eine Kunst für sich, die zudem beim Anwender Vorkenntnis über den *Aufbau* der angesprochenen Datenbank als gegeben voraussetzt.

Informationsvisualisierungssystemen zur Veranschaulichung von mitunter abstrakten Sachverhalten will diese Arbeit den Ansatz der sogenannten *Relevanzkugel-Metapher* vorstellen, die bisherige Entwicklung kurz zusammenfassen, und die Implementierung einer zur Einbettung auf Seiten des World Wide Web geeigneten solchen Relevanzkugel beschreiben. Als Sprachen kommen hierbei sowohl VRML zur Spezifikation der graphischen Elemente als auch Java zur Ablaufsteuerung und Anbindung an eventuelle weitere Programme (wobei wegen ihrer Verbreitung im World Wide Web primär an Java-Applets gedacht ist) zum Einsatz.

Inhaltsübersicht

Diese Arbeit besteht aus sechs hauptsächlichen Kapiteln mit jeweils unterschiedlichen Schwerpunkten sowie mehreren Anhängen:

- Kapitel 1 enthält eine kurze Einführung in die Thematik von Informationsretrieval und -visualisierung
- Kapitel 2 beschreibt das grundlegende Konzept des Relevanzkugel-Visualisierungswerkzeugs und dessen Vorgeschichte, liefert eine erste Einschätzung, und geht daneben kurz auf einige andere Visualisierungsansätze zur selben Problemstellung ein.
- Kapitel 3 stellt die bei der hier beschriebenen Implementierung zur Anwendung kommenden Sprachen und Schnittstellen im einzelnen vor.
- Kapitel 4 enthält neben einer kurzen Anforderungsanalyse primär Ideen und Gedanken zur technischen Umsetzung des grundlegenden Relevanzkugelkonzepts mit besonderem Augenmerk auf Verbesserungsmöglichkeiten in Bezug auf dessen potentielle Schwächen.
- Kapitel 5 beschreibt die letztendlich gewählte eigentliche Umsetzung.
- Kapitel 6 beschließt die Arbeit mit einer kurzen Analyse der vorgestellten Implementierung und dem Versuch, die Zukunftstauglichkeit des Relevanzkugelkonzepts einschließlich eventuell nachfolgender Versionen grob einzuschätzen.

- Anhang A beschreibt detailliert die einzelnen im Rahmen der Arbeit erstellten und verwendeten VRML-Knotenprototypen. Quellen- und Abbildungsverzeichnis schließlich finden sich in Anhang B bzw. Anhang C.

Kapitel 1 : Grundbegriffe

1.1. Information Retrieval

Information-Retrieval-Systeme dienen, wie der Name schon andeutet[1], als Schnittstelle zwischen dem Wissensbedürfnis des menschlichen Anwenders auf der einen und den in einer Datenbank tatsächlich enthaltenen Informationen auf der anderen Seite. Der Anwender drückt seine Wünsche durch mehr oder weniger gezielte Anfragen an das System aus, und dieses reagiert nach entsprechender Suche in der Datenbank mit der Ausgabe einer Menge von Informationen, die nach 'Ansicht' des Systems – in Form der mehr oder weniger komplexen systemeigenen Suchalgorithmen – die durch diese Anfragen gestellten Anforderungen möglichst gut erfüllen. Vom Standpunkt sowohl des Anwenders als auch des Entwicklers definiert sich hierbei 'möglichst gut' derart, daß das System einerseits möglichst viele (idealerweise alle) Informationen findet, die für den Anwender tatsächlich von Interesse sind, sich andererseits bei der Ausgabe aber auch so gut wie möglich auf nur diese beschränkt. Erschwert wird die Aufgabenstellung dadurch, daß nicht automatisch vorausgesetzt werden kann, daß der menschliche Benutzer seine Suche notwendigerweise mit einer präzisen und auf die verwendete Kombination von Retrieval-System und Datenbank zugeschnittenen Anfrage beginnt; es muß sowohl dem Fall Rechnung getragen werden, daß der Anwender zu Beginn noch gar nicht weiß, wonach er genau sucht, als auch demjenigen, daß er im Verlauf der Interaktion mit dem System auf Daten stößt, die völlig neue Informationsbedürfnisse wecken. Um diese Probleme zu lösen, gibt es eine Reihe unterschiedlicher Ansätze; im Rahmen dieses Kapitels soll dabei zur Illustration nur auf die zwei grundlegendsten Modelle genauer eingegangen werden.

- Am nach wie vor verbreitesten sind die sogenannten *Exact-Match*-Systeme, denen jeder Benutzer eines gängigen Datenbanksystems wahrscheinlich schon einmal begegnet ist. Diese Systeme sind dadurch charakterisiert, daß der Anwender eine mehr oder weniger detaillierte Informationsanfrage stellt und als Ausgabe eine Liste von Datenbankeinträgen zurückerhält, die genau die in der Anfrage ausgedrückten Anforderungen exakt erfüllen. Beim *booleschen Information-Retrieval* beispielsweise setzen sich Anfragen unter Verwendung der gängigen booleschen Operatoren UND, ODER, und NICHT aus einzelnen

[1] engl. 'retrieval' = 'Zurückholen'

Suchtermen und/oder Vergleichsausdrücken zusammen; die Ergebnismenge besteht aus allen Datensätzen, für die der Gesamtausdruck den Wert *wahr* annimmt. Das Grundproblem der Exact-Match-Systeme besteht darin, daß sie keinerlei weitergehendes Maß zur Ergebnisbewertung (etwa eine Anordnung nach irgendeinem Relevanzkriterium) anbieten: die Gesamtmenge aller in der Datenbank vorhandenen Informationen zerfällt durch jede Anfrage genau in die beiden disjunkten Teilmengen A solcher Daten, die diese Anfrage erfüllen und B derer, die es nicht tun. Es gibt keine weiteren Abstufungen, und jeder Datensatz in einer der beiden Teilmengen ist vom Standpunkt des prinzipiellen Ansatzes – nicht unbedingt von dem des Anwenders! – her betrachtet jedem anderen in derselben Menge genau gleichwertig. Andererseits findet man diese Systeme wegen ihres prinzipiell leicht verständlichen und implementierbaren Konzepts praktisch überall, was zu einer gewissen Gewöhnung führt und damit naturgemäß zu ihrer Akzeptanz beiträgt.

- Die *Best-Match*-Systeme, wiewohl prinzipiell als Erweiterung der Exact-Match-Methode entstanden, geben dagegen den Anspruch auf *exakte* Erfüllung der formulierten Anfrage auf und versuchen statt dessen mit Hilfe von internen Bewertungsfunktionen und der dem Anwender zusätzlich zur Verfügung gestellten Möglichkeit, Teilausdrücke der Anfrage individuell nach Relevanz zu gewichten, solche Informationen aufzuspüren und auszugeben, die dem Informationsbedürfnis des Benutzers nur *möglichst gut* gerecht werden. Es gibt zu diesem Zweck eine Anzahl verschiedener Ansätze und Denkmodelle, die allgemein auf der Idee aufbauen, zunächst zu bewerten, wie gut die Datensätze jeweils die einzelnen Teilanforderungen erfüllen und dann die Resultate in geeigneter Weise unter Einbeziehung der vom Anwender mitgegebenen Relevanzwerte so zu kombinieren, daß letztendlich eine Auswahl möglichst passender 'Treffer' vorgenommen werden kann. Die einzelnen Ausführungen dieser Grundidee (wie z.B. Ranking-, Vektorraum-, Clustering- oder probabilistische Modelle) unterscheiden sich prinzipiell lediglich durch die verwendeten exakten Gewichtungsfunktionen und die Organisation und Kombination der erhaltenen Einzelergebnisse. In der Regel werden dabei die internen Relevanzfaktoren der einzelnen Datensätze in Relation zu den Suchtermen durch mehr oder weniger feste und mitunter subjektive Regeln (beispielsweise über die Häufigkeit des Auftretens eines Stichworts innerhalb von in der Datenbank

enthaltenen Textdokumenten) einmal festgelegt und bleiben dann im alltäglichen Betrieb konstant.

Beiden Arten von Systemen gemeinsam ist das Problem, daß sich die Formulierung geeigneter Anfragen nicht immer einfach gestaltet. Je nach Art des Interesses können sich durch Kombination prinzipiell einfacher Einzelterme (und im Fall von Best-Match-Systemen zusätzlicher subjektiver Gewichte) Suchausdrücke von nahezu beliebiger Kompliziertheit ergeben; daneben ist eine gewisse Mindestvertrautheit mit dem gegebenen Datenbanksystem Bedingung, damit nicht durch ein Mißverständnis etwa bei der Auswahl von Stichworten interessante Informationen nicht erfaßt werden bzw. zu viel für den Anwender unwichtiger 'Datenmüll' ausgeworfen wird. Außerdem stehen die einzelnen Interaktionen quasi im leeren Raum; es ist allein Sache des Anwenders, sich durch sukzessives Umformulieren von Anfragen, auf die noch keine zufriedenstellende Reaktion erfolgt ist, an ein brauchbares Resultat heranzuarbeiten, da das System jede ihm gestellte Aufgabe unabhängig von eventuell ähnlichen früheren Vorgängen völlig neu angeht.

Entsprechend existieren neben diesen beiden Varianten weitere Ansätze für Retrieval-Systeme – darunter einige von gehöriger interner Komplexität im Interesse einfacherer Handhabung durch den *Benutzer* –, und die Forschungsarbeit ist noch lange nicht abgeschlossen. Als Beispiele genannt seien hier nur kurz die sogenannten *Relevance-Feedback*-Systeme, die über das Wechselspiel von Benutzeranfrage und Systemausgabe gewissermaßen Buch führen und auf dieser Basis die Relevanz einzelner Datenbankeinträge intern neu gewichten, sowie Modelle auf der Basis von neuronalen Netzen oder Methoden aus dem Bereich der künstlichen Intelligenz. Eine detailliertere Übersicht, als sie dieses Kapitel bieten kann, findet sich etwa in [Hemmje 99], Kapitel 3.

1.2. Informationsvisualisierung

Der Begriff *Visualisierung* bezeichnet die bildhafte Aufbereitung und Darstellung von Informationen – im modernen Kontext oft mit Hilfsmitteln der Computergraphik –, um deren Analyse, Verständnis, und Weitervermittlung zu erleichtern. Dabei unterscheidet die gängige Literatur (wie z.B. [Card 99] oder [Müller/Schumann 00]) im Allgemeinen grob zwischen einerseits der historisch älteren *wissenschaftlichen Visualisierung* zur Darstellung von primär physikalischen Daten und Zusammenhängen und andererseits der *Informationsvisualisierung* im engeren Sinn, die

prinzipiell dasselbe für abstraktere Informationen leistet. Das grundsätzliche Ziel ist in beiden Fällen gleichermaßen, die enge Verknüpfung zwischen den Funktionen der menschlichen Wahrnehmung und denen der menschlichen Kognition so auszunutzen, daß die letzteren entlastet und für wichtigere Aufgaben als die reine Aufnahme und Interpretation der anfallenden Informationen freigestellt werden; die Unterscheidung zwischen beiden Kategorien ist historisch gewachsen und basiert wirklich in der Hauptsache lediglich auf der *Herkunft* der abzubildenden Daten.

Die Idee, externe Hilfsmittel als Denk- und Verständnishilfen sowie Gedächtnisstützen heranzuziehen, ist an sich nicht neu. Ein grundlegendes Beispiel hierfür ist die Erfindung der Schrift, mit deren Hilfe Wissen gespeichert und weitergegeben werden kann, ohne sich allein auf das menschliche Gedächtnis und die mündliche Überlieferung zu verlassen; zudem kann ein geübter Leser schriftliche Informationen dank der größeren Bandbreite seiner visuellen Wahrnehmung schneller aufnehmen als gesprochene Worte, und das Medium erlaubt beliebig viele Wiederholungen nach Bedarf. Oder man denke an das sprichwörtliche Bild, das mehr aussagt als tausend Worte, wobei 'Bild' hier nicht notwendigerweise im Sinne von 'Kunst' verstanden werden soll; eine einfache Skizze oder ein Diagramm am richtigen Platz kann auf einen Blick Zusammenhänge veranschaulichen, die sonst einer längeren Erklärung und vielleicht sogar Diskussion bedürften. Der primäre Vorteil der modernen computerunterstützten Visualisierungstechniken liegt vielmehr in deren Möglichkeiten, unter Ausnutzung der Fortschritte auf den Gebieten der Computergraphik und Rechenleistung zunehmend komplexe Zusammenhänge und Abläufe sowohl dynamisch als auch interaktiv zu modellieren, bis hin zur vollständigen anschaulichen Echtzeitsimulation von komplizierten Vorgängen und Systemen.

Beim grundlegenden Visualisierungsvorgang unterscheidet die gängige Literatur meist vier logisch aufeinander aufbauende Schritte, die allerdings ebenfalls nicht immer streng linear aufeinander folgen müssen, da es innerhalb eines Einzelschritts oder auch zurückgreifend zur Bildung von Schleifen kommen kann; nichtsdestoweniger spricht man im allgemeinen von der sogenannten *Visualisierungspipeline*:

1.) Sammeln und Erfassen der Rohdaten.

2.) Einordnen besagter Rohdaten in ein geeignetes *Datenschema*, wobei durch (zum Beispiel) Fehlerkorrektur, Zusammenfassung, und/oder weitergehen-

de Interpretation der ursprünglichen Werte Information sowohl verlorengehen als auch hinzugewonnen werden kann.

3.) Erstellen eines *Visualisierungsschemas*, welches die Art der Darstellung der im vorigen Schritt aufbereiteten Daten festlegt.

4.) Erzeugen der eigentlichen Darstellung aus den aktuellen Daten anhand der obigen Schemata.

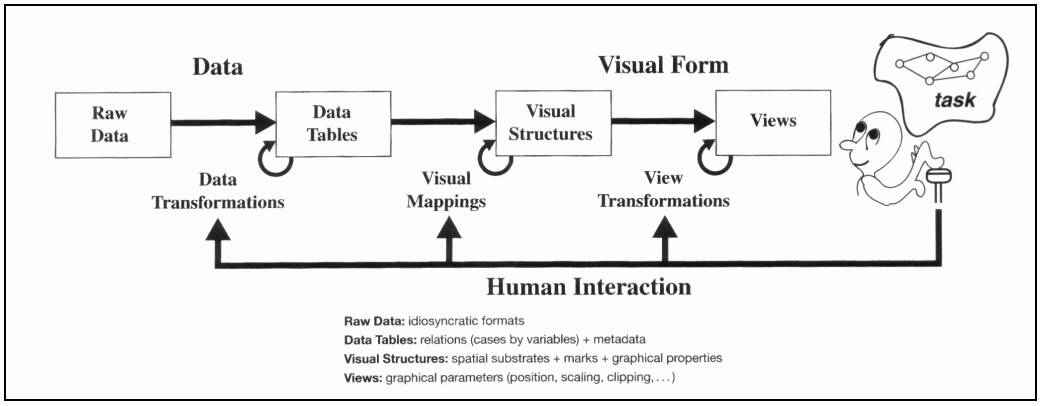

Abb. 1: Visualisierungsprozess nach [Card 99] (S. 17)

Die möglichen Aufgabenstellungen für Visualisierungssysteme sind vielfältig, ebenso ihre verschiedenen Spielarten; mit aus diesem Grund hat sich bis heute noch keine einheitliche Systematik zu ihrer Einordnung durchgesetzt. Zwei offensichtliche Möglichkeiten zur Kategorisierung von Visualisierungsansätzen sind dabei einerseits der zugrundeliegende Verwendungszweck, andererseits die Eigenschaften der verwendeten Darstellung an sich. Beispielsweise unterscheiden die Autoren von [Müller/Schumann 00] in ihrem Einführungskapitel in Bezug auf die Aufgaben solcher Systeme in der wissenschaftlichen Visualisierung, mit der sich das Buch hauptsächlich befaßt, zwischen *explorativer Analyse*, bei der in der vorhandenen Datenmenge nach Informationen und Strukturen gesucht wird, *konfirmativer Analyse* zur Überprüfung daraus formulierter Hypothesen, und schließlich *Präsentation* der erzielten Ergebnisse. [Card 99] bezieht sich in ähnlicher Weise, jedoch allgemeiner auf den Prozeß der sogenannten *Informationskristallisation* (*knowledge crystallization*), bei dem es ebenfalls darum geht, Informationen erst einmal zu finden, dann diese Informationen zum besseren Verständnis in ein geeignetes Schema einzuordnen, das gegebenenfalls erst einmal selbst gefunden werden muß, und schließlich das resultierende Endergebnis zu präsentieren oder

auf dessen Grundlage Entscheidungen zu treffen; wie in Abb. 2 gezeigt, läßt sich dieser Vorgang weiter in kleinere Einzelschritte aufteilen, die nicht unbedingt in jedem Fall alle zur Anwendung kommen und auch nicht unbedingt immer die gleiche Abfolge durchlaufen müssen. Visualisierungstechniken können dabei prinzipiell an jedem einzelnen Punkt zur Anwendung kommen.

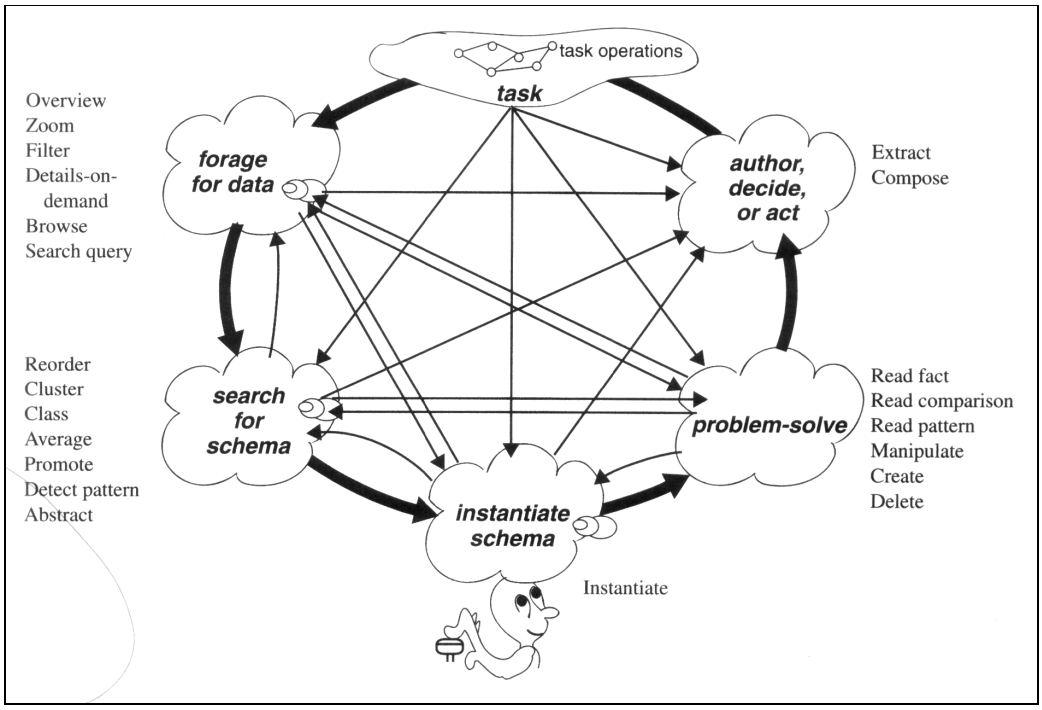

Abb. 2: Der Informationskristallisierungsprozeß nach [Card 99] (S. 10)

Zur näheren Katalogisierung eines Visualisierungssystems nach verwendeter *Darstellung* liegt es nahe, deren einzelne Bestandteile näher zu betrachten (Einteilung frei nach [Card 99], S. 26 - 33):

- Da ist zunächst einmal der zugrundeliegende *Raum* mit den in ihn eingebetteten *Bezugsachsen*. Die Darstellung auf einem handelsüblichen Computerbildschirm ist naturgemäß zweidimensional (Höhe mal Breite), wobei beide Achsen in diskrete Abschnitte aufgeteilt sind, durch deren Kombination sich die Positionen der einzelnen Bildpunkte bzw. Pixels ergeben. Konzeptionell jedoch kann das grundlegende Schema durchaus mehr oder weniger als zwei Achsen enthalten, die auch nicht in konventioneller Weise rechtwinklig zueinander angeordnet sein müssen und deren Aufteilung anders aussehen kann. Beispielsweise könnte ein und dieselbe Achse mehrfach wiederholt auftreten,

wobei ein Exemplar eine Gesamtübersicht bietet und die weiteren progressiv mehr ins Detail gehen, indem sie einen Ausschnitt der vorhergehenden Abbildung vergrößern.

- Dann sind da natürlich die diversen Markierungen, Symbole, oder Objekte, die anhand der eben definierten Bezugsachsen positioniert werden. Das Spektrum reicht hierbei von einfachen Punkten, die nur ihre jeweilige Position markieren (auch wenn die eigentliche Markierung naturgemäß zwei- oder dreidimensional sein muß, um überhaupt sichtbar zu sein) bis hin zu Flächen mit Informationsgehalt oder komplexen Objekten, die bei Bedarf eigene Interaktionsmöglichkeiten anbieten. Daneben können Beziehungen zwischen den Einzelsymbolen z.B. durch Verbindungslinien oder Einschließen eines Symbols in einem anderen modelliert werden.

- Interessant sind auch die positions*unabhängigen* Eigenschaften der einzelnen Symbole, wie Größe, Form, Orientierung, Farbe, Helligkeit bzw. Graustufen, und Textur. Mit ihrer Hilfe lassen sich Zusatzinformationen vermitteln, die aus der reinen Anordnung der Symbole im Raum nicht eindeutig hervorgehen; beispielsweise wird ein Betrachter in einem Punktdiagramm, dessen einzelne Positionsmarkierungen von unterschiedlicher Größe sind, in der Regel die größeren Symbole als wichtiger empfinden. Freilich erreicht man bei Verwendung dieser Eigenschaften normalerweise nicht dieselbe Trennschärfe in Bezug auf Details, wie sie sich durch geschickte Positionierung vermitteln läßt; das menschliche Auge unterscheidet leichter zwischen zwei nahe beieinanderstehenden Objekten als zum Beispiel zwischen zwei annähernd gleichen Farbtönen[1]. Außerdem ist eine gewisse Mindestsymbolgröße gerade bei der computergestützen Visualisierung Bedingung, wenn die zu vermittelnde Zusatzinformation nicht durch Zusammenschrumpfen des Symbols auf ein oder wenige Pixel verlorengehen oder zumindest stark eingeschränkt werden soll.

- Zuletzt bleiben noch die zeitlichen Abläufe und Interaktionsmöglichkeiten zu bedenken. Da die menschliche Wahrnehmung sehr sensibel auf Veränderungen reagiert, kann die Aufmerksamkeit des Betrachters durch geeignete Bewegungsabläufe in der Darstellung leicht angezogen und gelenkt werden.

[1] Obendrein sollte gerade bei der Verwendung von Farbkodierungen die Möglichkeit von Sehschwächen des Betrachters immer bedacht werden.

Desgleichen lassen sich dynamische Prozesse praktisch nur auf diese Weise anschaulich darstellen. Auf der anderen Seite kann der Betrachter allerdings auch durch unglücklich eingesetzte Animationen sehr leicht vom Zweck der Darstellung *abgelenkt* werden. – Interaktion ist im Prinzip zunächst nichts anderes als eine Veränderung der Darstellung als Reaktion auf eine Eingabe des Betrachters, und für sie gilt daher weitgehend das gleiche. *Gleichzeitig* bietet sie aber auch eine völlig neue Dimension zum Vermitteln von Verständnis; schon eine einfache Möglichkeit zum Abruf von Detailinformationen durch 'Antippen' einzelner Szenenbestandteile (Location Probing) oder zur Änderung der virtuellen eigenen Position in einer dreidimensionalen Szene eröffnet völlig neue Perspektiven und trägt dem menschlichen Bedürfnis Rechnung, etwas Interessantes bei Bedarf aus verschiedenen Richtungen betrachten und möglichst eingehend untersuchen zu können.

Kapitel 2 : Das Relevanzkugelmodell

Die *Relevanzkugel* nun ist ein schon etwas länger existierendes Visualisierungswerkzeug auf Punktdiagrammbasis zum Zweck der intuitiven Darstellung komplexer Mehrfachrelevanzbeziehungen und der Vorselektion in potentiell großen Informationsmengen. Das heißt, man hat eine Situation, in der eine Menge von Dokumenten oder Datensätzen einer Menge von Referenztermen, beispielsweise Suchbegriffen, gegenüberstehen, die für die einzelnen Dokumente mehr oder weniger relevant sind, so daß sich – für *m* Referenzterme und *n* Datensätze – eine Relevanzmatrix mit insgesamt *m* x *n* Einträgen ergibt. Für kleine Werte von *m* und *n* läßt sich diese Matrix natürlich auch leicht in Tabellenform fassen. Selbst für größere Werte ist eine entsprechende Darstellung noch denkbar, solange die Auswahl an einzelnen Relevanzwerten beschränkt ist, wie beispielsweise für den grundlegenden Fall, daß einfach nur zwischen 'relevant' und 'nicht relevant' unterschieden wird; dann können nämlich die einzelnen Werte leicht durch unterschiedliche Zahlen, Symbole, Farben oder Ähnliches abgebildet werden. Dieser Ansatz stößt jedoch da an seine Grenzen, wo für die Relevanzwerte eine nichttriviale Menge von Einzelmöglichkeiten oder gar Werte aus einem echten Kontinuum (wie beispielsweise den reellen Zahlen aus dem Intervall [0,1]) in Frage kommen, da die Darstellung schnell zu unübersichtlich wird, um noch größere Zusammenhänge erfassen zu können. An diesem Punkt nun kommen komplexere Modelle wie die Relevanzkugel ins Spiel.

2.1. Modellbeschreibung

Kennzeichnend für die Relevanzkugeldarstellung ist der einfache Aufbau aus drei klar getrennten Einzelkomponenten (Abb. 3):

- Zum einen der Innenraum der Kugel selbst, der normalerweise durch eine angedeutete oder explizit dargestellte transparente Schale eingeschlossen und so konzeptionell von der 'Außenwelt' getrennt wird. Diese Schale darf dabei den Zugriff des Betrachters auf die im Kugelinneren bzw. auf der anderen Seite der Kugel befindlichen Objekte, die die anderen beiden Bestandteile der Darstellung bilden, nicht blockieren, da die Interaktion mit allen Komponenten zentraler Bestandteil des Modells ist.

- Dazu kommen die auf der Kugelschale verteilten sogenannten *Referenzobjekte*. Diese können vom Benutzer auf der Kugeloberfläche frei verschoben

werden und sind zudem mit einem ebenfalls vom Benutzer regelbaren Gewichtsfaktor zwischen 0 und 1 behaftet. Konzeptionell entspricht jedes Referenzobjekt einem Referenzterm der Relevanzmatrix.

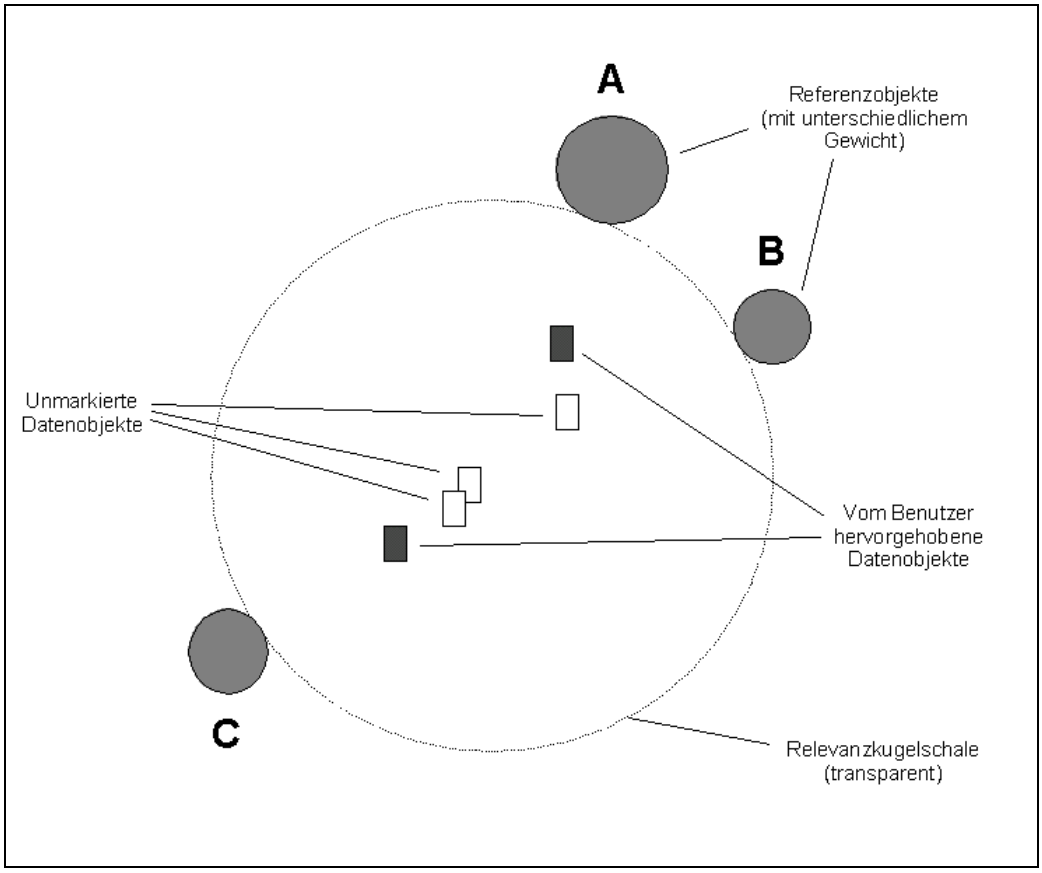

Abb. 3: Die Komponenten der Relevanzkugel im Überblick

- Im *Inneren* der Relevanzkugel schließlich findet man die sogenannten *Datenobjekte*, die die eigentlichen Dokumente bzw. Datensätze symbolisieren. Jedes Datenobjekt hat seine eigenen charakteristischen Relevanzfaktoren, die angeben, wie hoch der Beitrag jedes einzelnen Referenzobjekts zu seiner endgültigen Position bewertet werden soll; die Verteilung der Datenobjekte in der Kugel ergibt sich dann aus der Verteilung und den Gewichten der einzelnen Referenzobjekte auf der Kugelaußenseite einer- und aus eben diesen Relevanzfaktoren andererseits, und zwar in Form einer gewichteten Vektoraddition. Daraus folgt, daß 'verwandte' Objekte, d.h. solche mit ähnlichen Relevanzwerten, immer näher beieinander stehen als solche, die eher wenig mitein-

ander gemein haben[1]. Ändert sich die Position oder das Gewicht eines Referenzobjekts, so ändern sich auch (in Echtzeit) die Datenobjektpositionen in Abhängigkeit von den jeweiligen Relevanzbeziehungen mehr oder weniger stark. – Zusätzlich zu dieser indirekten Möglichkeit, auf die Datenobjekte einzuwirken, sieht das Relevanzkugelkonzept auch die Option vor, einzelne Datenobjekte zur leichteren Unterscheidung auf geeignete Weise, zum Beispiel farblich, hervorzuheben. Auf diese Weise kann der Anwender beispielsweise eine Handvoll potentiell interessanter Objekte auswählen und gezielt beobachten, wie sie sich bei weiterer Manipulation an den Referenzobjekten im Einzelnen verhalten. Klassischerweise werden die Referenzobjekte zur Unterscheidung mit Namen versehen, während bei den Datenobjekten der besseren Übersichtlichkeit halber die reine Symboldarstellung bevorzugt wird; das führt allerdings dazu, daß die *Identifizierung* der einzelnen Datenobjekte anderweitig, beispielsweise durch Kopplung an eine kugelexterne eigene Anzeigemöglichkeit, gehandhabt werden muß.

Neben den bereits genannten Möglichkeiten zur Manipulation einzelner Objekte steht dem Betrachter zusätzlich die Option zur Verfügung, die Kugel als Ganzes zu drehen und damit aus den verschiedensten Blickwinkeln zu studieren. Außerdem können zu den drei Kernbestandteilen des Konzepts implementierungsabhängig weitere Elemente kommen, um beispielsweise zusätzliche Informationen einzubauen oder dem Betrachter das Studium der Kugel als Ganzes zu erleichtern. Solche Zusatzelemente sind natürlich nicht in derselben Weise zentrale Teile des Relevanzkugelmodells wie eben Referenzobjekte, Datenobjekte und Kugelschale; nichtsdestoweniger können sie sich für eine gegebene Implementierung als nützlich erweisen und eventuell Ideen zur Erweiterung des Konzepts liefern.

2.2. Geschichte

Ähnlich, wie sich das Relevanzkugelmodell aus anderen Ansätzen – insbesondere dem in 2.5.3. beschriebenen VIBE-Modell – herleiten läßt, hat auch die im Rahmen dieser Arbeit zu erstellende praktische Umsetzung des Konzepts Vorgänger, auf deren Erfahrungen aufgebaut werden kann. Eine frühe Implementierung findet sich

[1] Natürlich ist es dazu nötig, daß die Relevanzwerte auf geeignete Weise aus den zugrundeliegenden Rohdaten extrahiert wurden und tatsächlich entsprechende Aussagekraft besitzen.

beispielsweise schon 1994 im Rahmen des *Lyberworld*-Projekts. Der Name ‚Lyberworld' ist dabei eine künstliche Verschmelzung der Begriffe ‚Library', ‚Cyber', und ‚World'; das System dient zur intuitiven Darstellung von Zusammenhängen innerhalb einer Datenbank aus einzelnen Dokumenten mit Hilfe dreier spezieller Werkzeuge, die nach einer erfolgreichen Stichwortsuche einzeln über ihre Symbole auf dem Bildschirm angewählt und zur besseren Visualisierung eben dieser Zusammenhänge zwischen dem Suchbegriff einerseits und gefundenen Dokumenten und eventuellen verwandten Begriffen andererseits eingesetzt werden können.

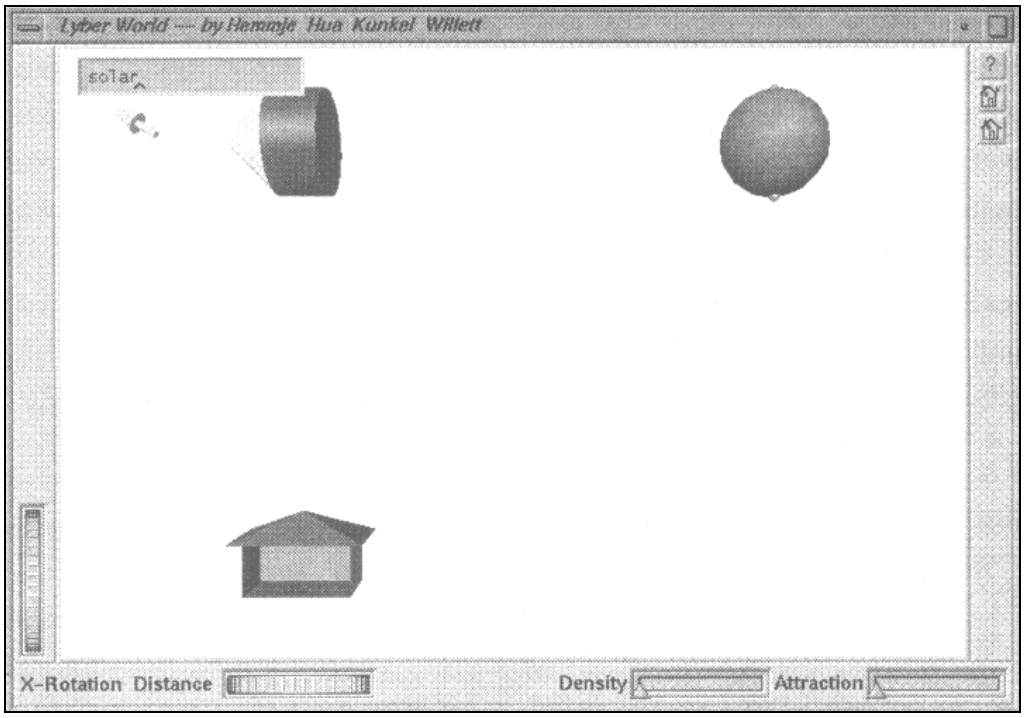

Abb. 4: Der LyberWorld-Startbildschirm ([Leissler 97], S. 61)

Diese Werkzeuge sind im einzelnen:

1.) Der Kontextbaum (*LyberTree*) mit alternierenden Ebenen für Dokumente und Suchbegriffe als Anzeigemöglichkeit für die Gesamtmenge der Dokumente, die durch einen gegebenen Suchbegriff angesprochen werden, bzw. umgekehrt der Suchbegriffe, die auf ein bestimmtes Dokument zutreffen. Die Darstellung erfolgt dabei in Form von einzelnen beschrifteten ‚Längsstreifen' entlang frei rollbarer waagerechter Zylinder, wobei durch die Auswahl einer solchen Facette ein vollständig neuer Zylinder der jeweils anderen ‚Ebene' (Dokumentenebene für angewählte Suchbegriffe, Suchbegriffebene für

selektierte Dokumente) quasi aufgeklappt und solange wie nötig beibehalten werden kann, woraus sich die Baumstruktur ergibt. (Abb. 5 und Abb. 6)

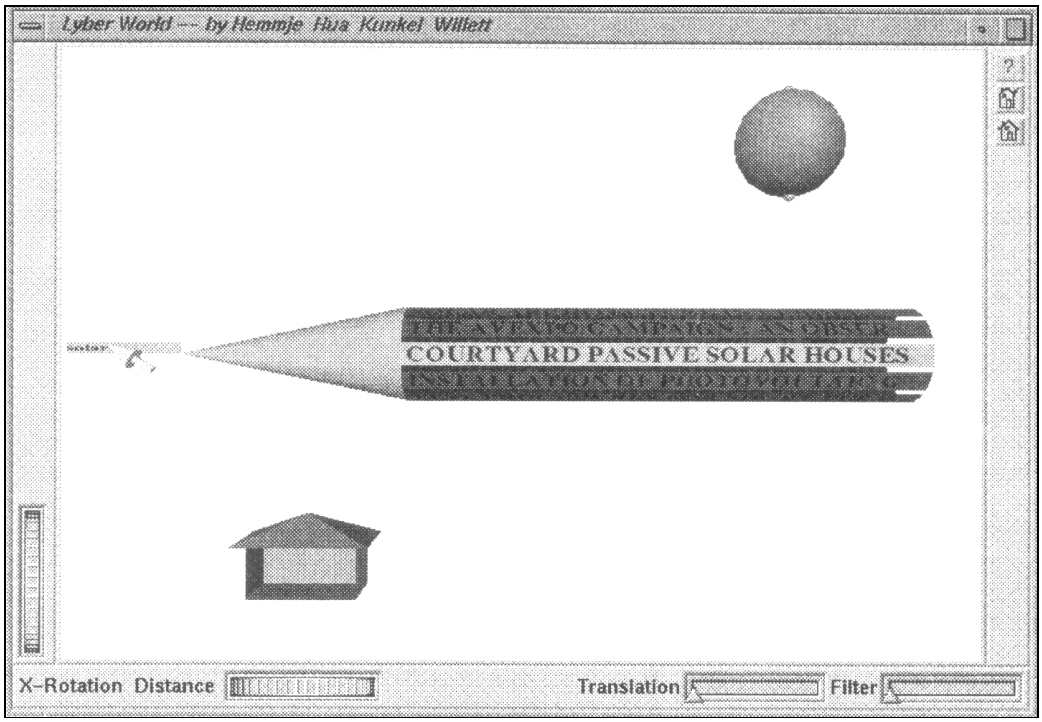

Abb. 5: Der LyberWorld-Kontextbaum ([Leissler 97], S. 63)

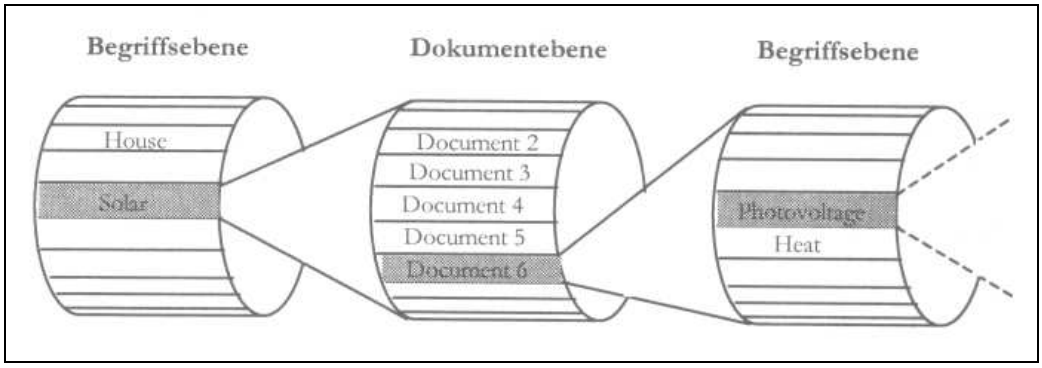

Abb. 6: Die Ebenen des Kontextbaums im Wechsel ([Leissler 97], S. 65)

2.) Die Lyberworld-Version der Relevanzkugel (*LyberSphere*, Abb. 7) zur genaueren Darstellung der Zusammenhänge zwischen ausgewählten Suchbegriffen und Dokumenten, insbesondere unter Einbeziehung von Gewichts- und Relevanzfaktoren, die im Kontextbaum konstruktionsbedingt nicht zur Geltung kommen können.

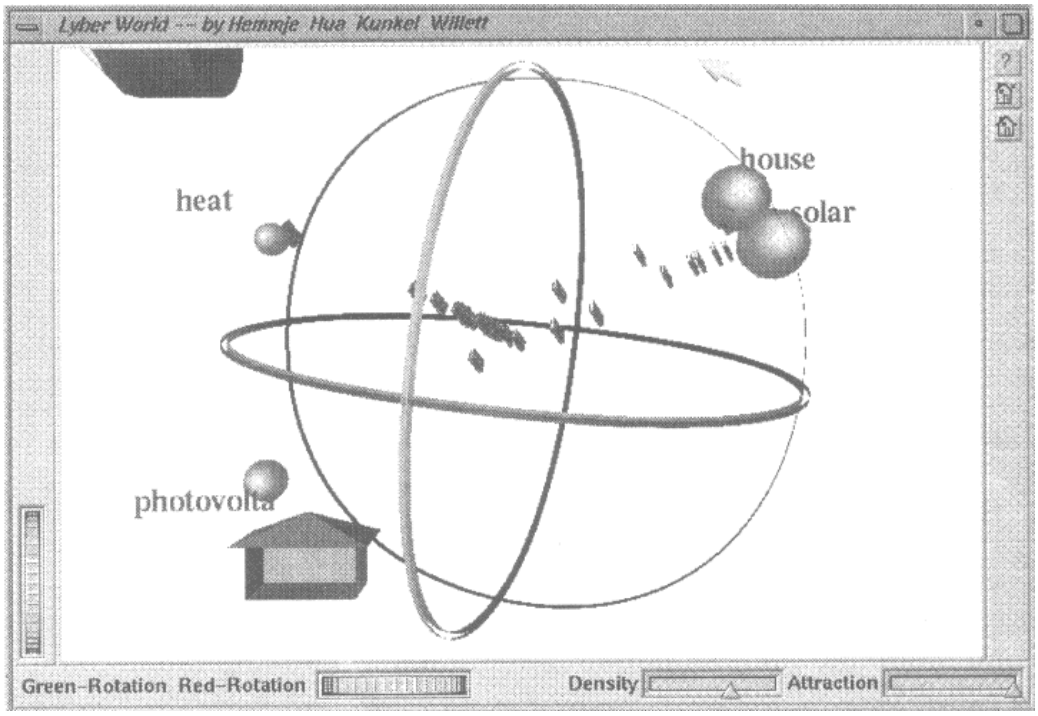

Abb. 7: Die LyberWorld-Relevanzkugel ([Leissler 97], S. 66)

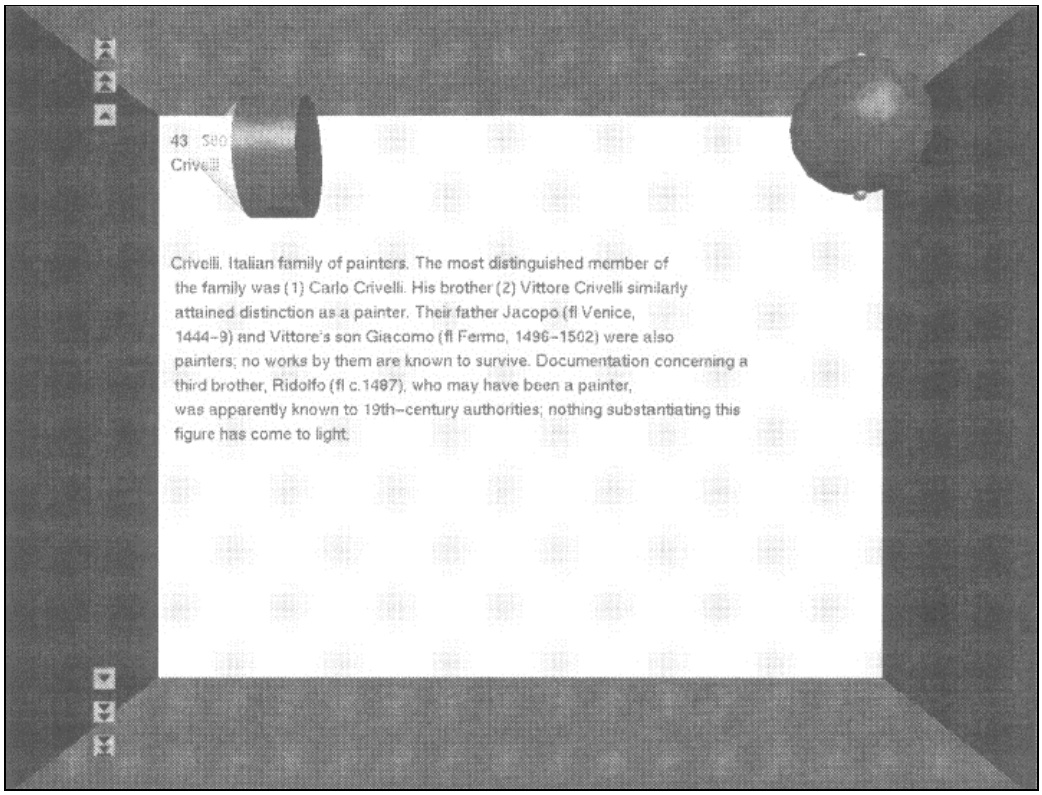

Abb. 8: Der LyberWorld-Dokumentenraum ([Leissler 97], S. 68)

3.) Der Dokumentenraum (*LyberRoom*), bei dem es sich im Wesentlichen nur um eine Anzeige für den Inhalt eines ausgewählten Dokuments handelt (Abb. 8). Allerdings kann im Gegensatz zu den beiden anderen Werkzeugen, von denen immer nur das eine oder das andere aktiv sein kann, der Dokumentenraum simultan mit entweder Kontextbaum oder Relevanzkugel verwendet werden, wobei sich dann die Darstellungen überlappen.

Neben einigen technischen Problemen lag die primäre Schwäche der ersten zur Anwendung kommenden Relevanzkugel in ihrer umständlichen Bedienbarkeit. Beispielsweise konnte ein Referenzobjekt zwar mit dem Mauszeiger angewählt, aber nur in Schritten fester Größe vermittels der Cursortasten auf der Tastatur des Rechners über die Kugeloberfläche bewegt werden; dasselbe galt für Drehungen der ganzen Kugel, wenn gerade kein Referenzobjekt ausgewählt war. Auch um das Gewicht und damit die Anziehungskraft eines Referenzobjekts zu verändern, mußte das betreffende Objekt erst einmal selektiert und dann sein Gewicht über einen separaten Schieberegler (Abb. 7, 'Attraction') neu eingestellt werden.

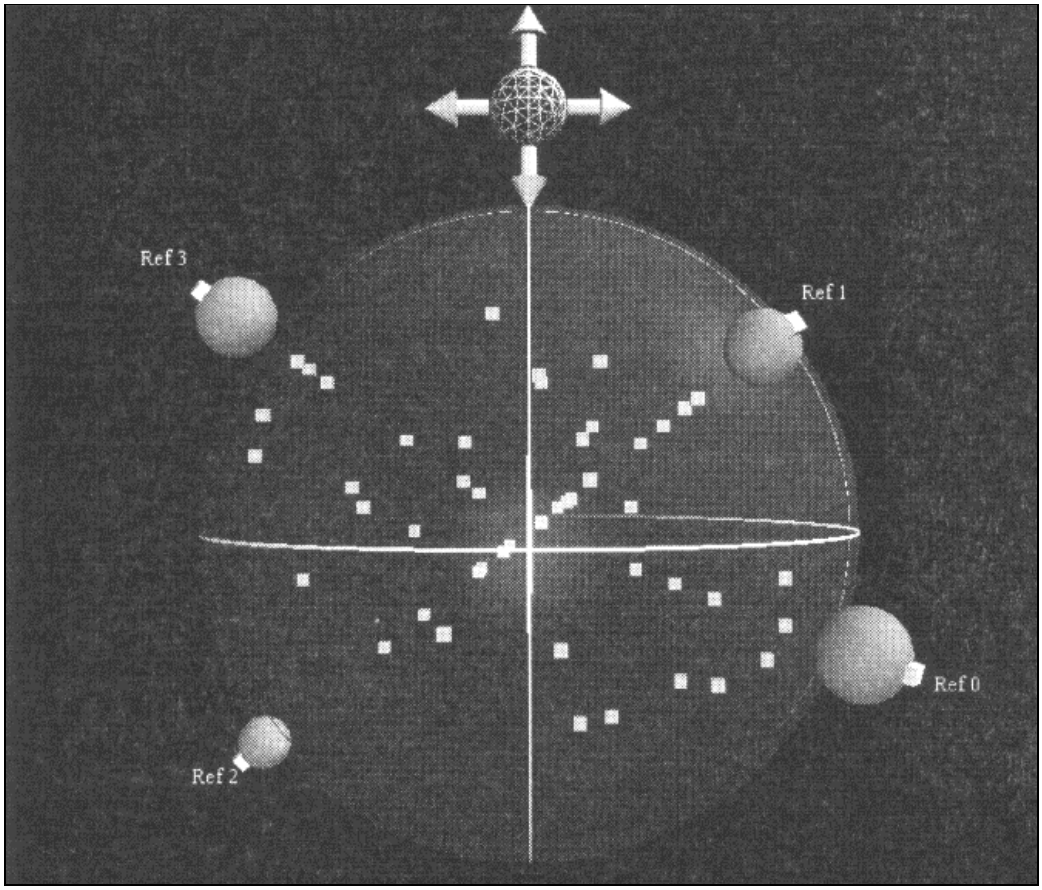

Abb. 9: Die verbesserte Relevanzkugel ([Leissler 97], S. 87)

In [Leissler 97] wird daher eine wesentlich verbesserte Implementierung des Konzepts vorgestellt. (Abb. 9) Diese basiert auf dem von der Firma Silicon Graphics entwickelten Open Inventor-Grafiksystem (und damit letztlich auf der bekannten OpenGL-Grafikschnittstelle derselben Firma) und behebt neben den technischen Schwierigkeiten der älteren Version auch den oben angesprochenen Kritikpunkt, indem die Kugelelemente *direkt* mit dem Mauszeiger erfaßt und manipuliert werden können. Zur besseren Handhabung der Kugel als Gesamtheit kommt außerdem ein neues Element, nämlich der sogenannte Relevanzkugelmanipulator (Abb. 10), der es erlaubt, die Relevanzkugel wie an einem Griff mit dem Mauszeiger zu erfassen und zu verschieben bzw. durch Drehung des kugelförmigen Mittelteils des Manipulators analog rotieren zu lassen.

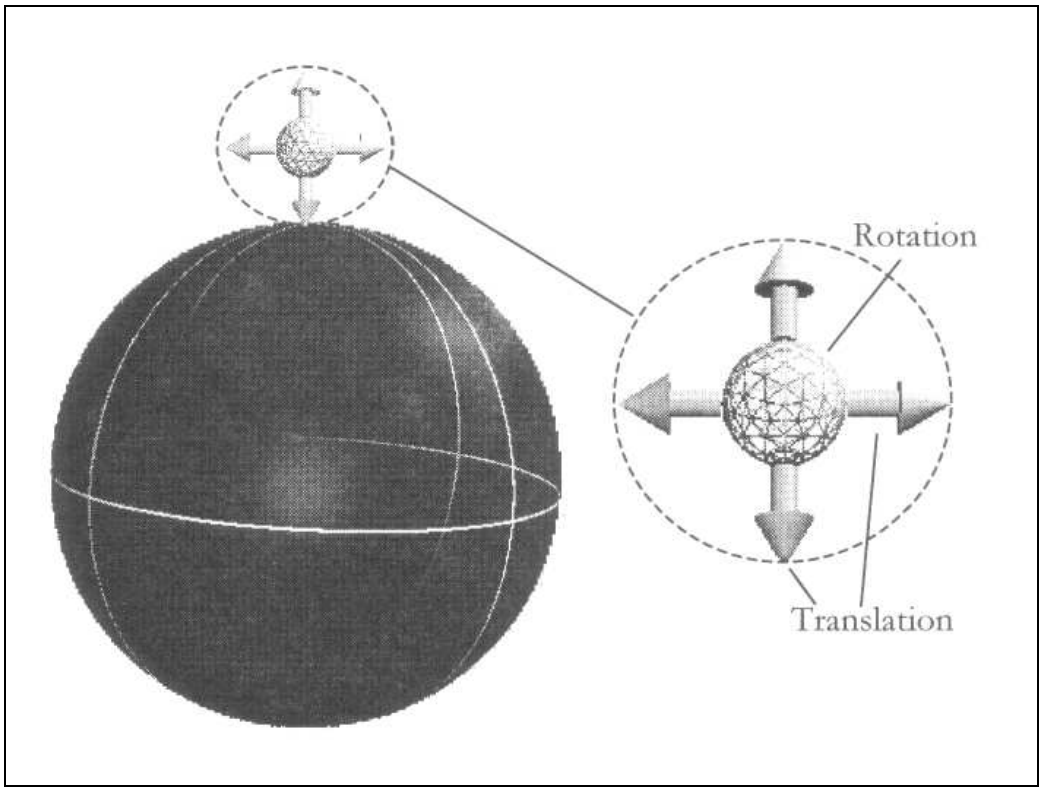

Abb. 10: Der Relevanzkugelmanipulator im Detail ([Leissler 97], S. 82)

Auch diese verbesserte Version der Relevanzkugel hat allerdings einen Nachteil mit ihrem Vorgänger gemeinsam: beide wurden speziell für eine vorgegebene Systemumgebung entwickelt (teilweise unter Verwendung proprietärer Software) und sind damit nicht ohne weiteres auf andere Umgebungen übertragbar. Im Interesse einer allgemeineren Verwertbarkeit des Relevanzkugelwerkzeugs ist aber eine möglichst

weitgehende Plattformunabhängigkeit wünschenswert. Hier setzt diese Arbeit mit dem Versuch an, eine Implementierung der Relevanzkugel unter Verwendung möglichst frei verfügbarer und standardmäßig systemunabhängiger Werkzeuge für das heutzutage praktisch allgegenwärtige World Wide Web zu erstellen.

2.3. Einordnung

Die prinzipielle Aufgabenstellung, für die das Relevanzkugelwerkzeug entworfen wurde, besteht in der Selektion von für den Betrachter potentiell interessanten Informationsobjekten aus einer größeren, im Regelfall eher unstrukturierten Menge, also gewissermaßen im Gewinnen eines Überblicks über die Gesamtsituation und dem Treffen einer Vorauswahl, auf deren Grundlage dann eine genauere Exploration der gefundenen Teilmenge an Informationen erfolgen kann. Damit fällt die Relevanzkugel primär in die Kategorie der Informations*such*werkzeuge; ein Herausarbeiten von Zusammenhängen ist ansatzweise durch Manipulation der Referenzobjekte und Beobachtung des daraus resultierenden Datenobjektverhaltens möglich, geht aber ohne weitere Hilfsmittel nicht ins Detail, und zur *Präsentation* von Informationen schließlich ist das Modell mangels über die eher abstrakte Objektrepräsentation hinausgehenden Möglichkeiten effektiv bestenfalls in Ausnahmefällen geeignet.

Neben dieser ersten Einordnung nach Funktionalität bietet es sich an, das Relevanzkugelmodell noch unter zwei weiterern Gesichtspunkten mit anderen Ansätzen zu vergleichen:

- *Historisch* leitet sich das Konzept nach [Leissler 97] und [Hemmje 99] vom hier später in Abschnitt 2.5.2. besprochenen VIBE-System ab, indem das Modell in die dritte Dimension erweitert wird und manipulierbare Referenzobjekte an die Stelle fixer Referenzpunkte treten.

- Vom zugrundeliegenden *Prinzip* her kann man die Relevanzkugel aber auch als Versuch interpretieren, ein normales dreidimensionales Punktdiagramm zu erweitern, indem potentiell zusätzliche Bezugsachsen eingefügt werden, ohne daß entsprechende weitere Dimensionen hinzukommen, und zusätzlich die Bezugsachsen zur Laufzeit dreh- und streck- bzw. stauchbar angelegt werden. Legt man diese Bezugsachsen nämlich so, daß sie genau durch den Kugelmittelpunkt und je ein Referenzobjekt verlaufen, und interpretiert man das Gewicht des Referenzobjekts als Skalierungsfaktor für die zugehörige Achse,

dann ergeben sich die Positionen der Datenobjekte in dem aufgespannten Koordinatensystem ganz analog zum konventionellen Diagramm nur noch aus den einzelnen Relevanzwerten; die Datenobjekte selbst stehen gewissermaßen fest und der Benutzer verändert durch Manipulation der Referenzobjekte lediglich das Bezugssystem. – Natürlich führt die Verwendung von mehr Bezugsachsen, als Dimensionen zur Verfügung stehen, sofort dazu, daß es auch mehr als nur einen gültigen Satz von Koordinaten für ein und denselben Punkt im Raum gibt und die Eindeutigkeit der Zuordnung verloren geht. Die dem Betrachter zusätzlich gebotenen Möglichkeiten, die einzelnen Achsen zu bewegen und zu skalieren, dienen in erster Linie dazu, eben diesen Nachteil durch die Herstellung eines sichtbaren Zusammenhangs zwischen Manipulation eines Referenzobjekts und den Reaktionen der Datenobjekte wieder auszugleichen.

2.4. Erste Einschätzung

Das Relevanzkugelmodell zeichnet sich in erster Linie durch seinen einfachen und vergleichsweise übersichtlichen Aufbau aus, zumal sich prinzipiell alle nötigen Bedienungselemente direkt in die Bestandteile der Kugel integrieren lassen. Es ist außerdem grundsätzlich sehr allgemein einsetzbar, da ein Datenobjekt so ziemlich alles Mögliche repräsentieren kann.

Eine Schwäche, die das Modell prinzipbedingt mit anderen Systemen auf Punktdiagrammbasis teilt, besteht darin, daß sich nie ganz ausschließen läßt, daß eventuell zwei (oder mehr) an sich verschiedene Datenobjekte dieselbe Position zugeteilt bekommen und sich so überlappen, daß eine saubere Trennung und Ansprache der einzelnen Objekte schwierig bis unmöglich wird. Dies kann eher zufällig geschehen, wenn etwa die Addition von eigentlich verschiedenen Teilvektoren zum selben Ergebnis führt; in diesem Fall kann der Konflikt leicht durch Manipulation geeigneter Referenzobjekte wieder aufgelöst werden. Schwieriger wird es, wenn die zur aktuellen Darstellung der Kugel herangezogenen Relevanzwerte[1] beider Objekte im Rahmen der Darstellungsgenauigkeit übereinstimmen. In diesem Fall bleibt eigentlich nur, dem Benutzer das aufgetretene Problem nach Möglichkeit geeignet anzuzeigen und Reaktionsmöglichkeiten anzubieten. – Potentiell verbesserungsfähig sind

[1] Das bedeutet nicht unbedingt a*lle* Relevanzwerte – lediglich die, deren zugeordnete Referenzobjekte aktuell ein Gewicht sichtbar größer als 0 haben.

außerdem die Identifizierungsmöglichkeiten für einzelne Datenobjekte und die Darstellung der Assoziation zwischen Referenz- und Datenobjekten, die im zugrundeliegenden Modell praktisch nur über die Datenobjektpositionen und deren dynamische Änderung bei Manipulationen an den zugeordneten Referenzobjekten ausgedrückt wird; allerdings wäre es ebenso kontraproduktiv, wenn durch solche Verbesserungsversuche die Übersichtlichkeit der Darstellung verlorengehen würde.

2.5. Verwandte Arbeiten

Dieser Abschnitt betrachtet kurz einige alternative Visualisierungsansätze, die ebenso wie das Relevanzkugelmodell primär dazu dienen, dem Benutzer einen guten Überblick über eine weitgehend beliebige Informationsmenge zu bieten und ihm zu ermöglichen, in dieser die für seine Bedürfnisse relevanten Einzelinformationen auf schnelle und einfache Weise zu finden. Diese Aufzählung erhebt bei weitem keinen Anspruch auf Vollständigkeit; dazu existieren heute zu viele verschiedene Informationsvisualisierungssysteme, die teilweise verschiedenste Darstellungsmethoden kombinieren und deren Namen mitunter außerhalb der Institute, die sie entwickelt haben, kaum bekannt sind.

2.5.1. GUIDO

Das *Graphical User Interface for Data Organization* (GUIDO) von Korfhage ([Hemmje 99], S. 33-34) steht hier als Beispiel für ein grundlegend einfaches Visualisierungsmodell auf Punktdiagrammbasis. Eine Anzahl von Vektoren bzw. Achsen, die einzelne Referenzterme (grob analog den Referenzobjekten der Relevanzkugel) repräsentieren, spannen einen Raum mit einer entsprechenden Zahl von Dimensionen auf, während die Objekte der Informationsmenge als Menge von Punkten dargestellt werden, deren Positionen sich direkt aus den Relevanzen der den einzelnen Achsen zugeordneten Referenzterme für das jeweilige Objekt ergeben. Abb. 11 illustriert ein Beispiel mit zwei Bezugsachsen.

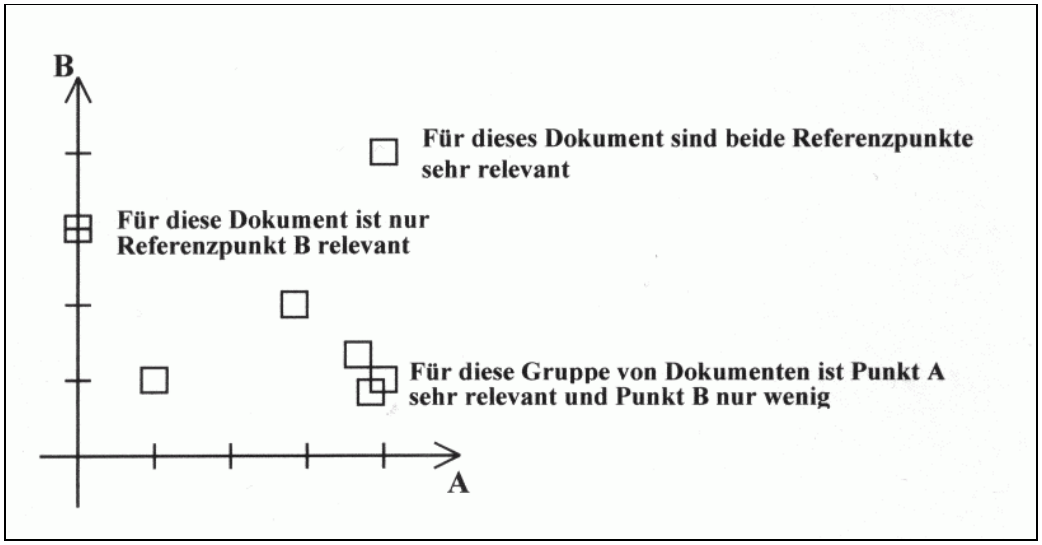

Abb. 11: GUIDO-Visualisierung mit zwei Referenztermen ([Hemmje 99], S.33)

Diese Darstellung ist leicht einsichtig und – rein mathematisch – einfach auf beliebig viele Dimensionen erweiterbar. In der Praxis verliert sie jedoch mit zunehmender Dimensionszahl schnell an Anschaulichkeit, da der menschliche Betrachter nur bis zu drei Raumdimensionen gewohnt ist (die obendrein zur Abbildung im Normalfall auf eine zweidimensionale Fläche projiziert werden müßten) und keine weitergehenden Interaktionsmöglichkeiten zur Verfügung stehen. Daher eignet sich dieser Ansatz in der Praxis nur für Anwendungen, bei denen die Anzahl der Referenzterme *mit Sicherheit* hinreichend klein bleibt.

Trotz dieser Einschränkung ist diese Art der Visualisierung mit dem Relevanzkugelkonzept bei genauerer Betrachtung recht nahe verwandt. Die Positionen der Datenobjekte in der Kugel ergeben sich im Prinzip nach genau derselben Rechenvorschrift wie hier, nur daß die Bezugsachsen dabei nicht starr im Raum fixiert sind, sondern vom Betrachter selbst wie in 2.3. beschrieben um den Ursprungspunkt gedreht und in Grenzen frei skaliert werden können. (Genaugenommen stehen auch die Datenobjekte in der Relevanzkugel die ganze Zeit über fest an ihren Positionen – nur eben relativ zu den veränderlichen Achsen!)

2.5.2. Bead

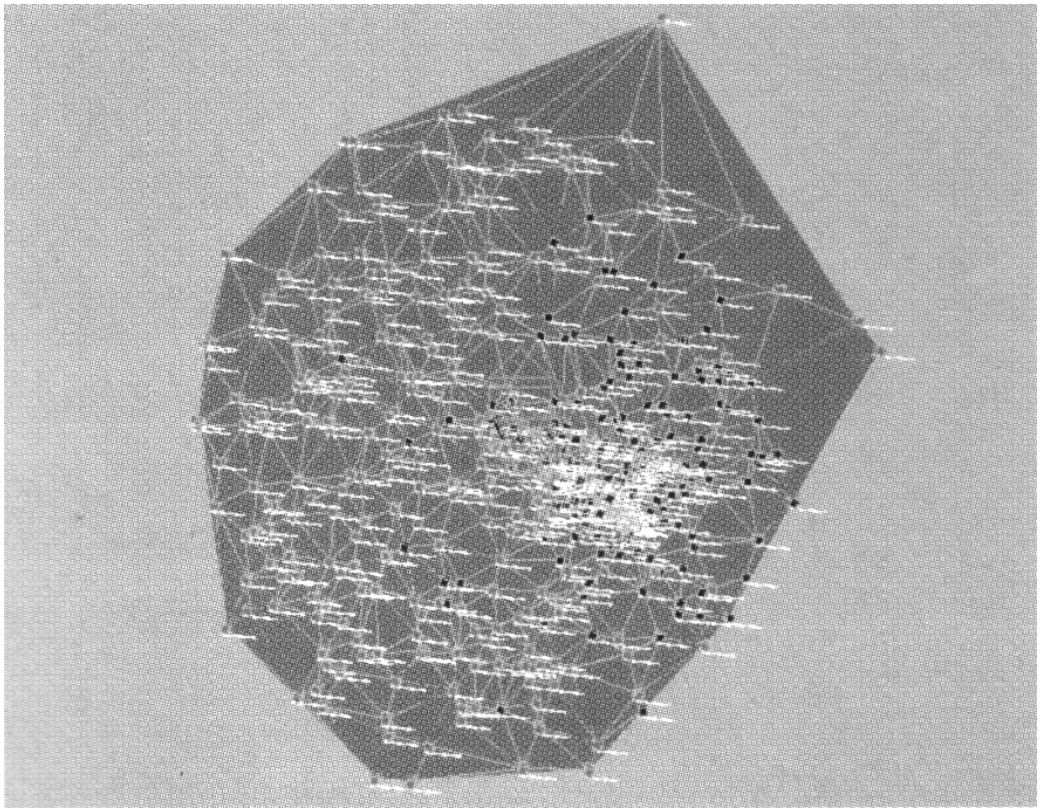

Abb. 12: Das Bead-System ([Chen 99], S.107)

Das Bead-System, erstmals 1992 von Chalmers und Chitson beschrieben, bestimmt den Verwandtschaftsgrad zwischen Dokumenten über die Anzahl und Häufigkeit gemeinsamer Teilausdrücke und modelliert dann auf dieser Basis ein komplexes System von anziehenden und abstoßenden Federkräften, um den einzelnen Dokumenten entsprechende Symbole in einer Partikelwolke bzw. bei späteren Versionen in einer übersichtlicheren Informationslandschaft so zu verteilen, daß verwandte Dokumente möglichst nahe beieinander positioniert werden. Die primären Interaktionsmöglichkeiten für den Anwender bestehen in der Bewegung durch die Darstellung, dem Hervorheben von zu bestimmten im Dialog ausgewählten Begriffen passenden Dokumentsymbolen, und in der Anforderung von Zusatzinformationen zu einem gegebenen Symbol bei Bedarf; die Verteilung der Symbole im Raum selbst ist statisch.

2.5.3. VIBE und VR-VIBE

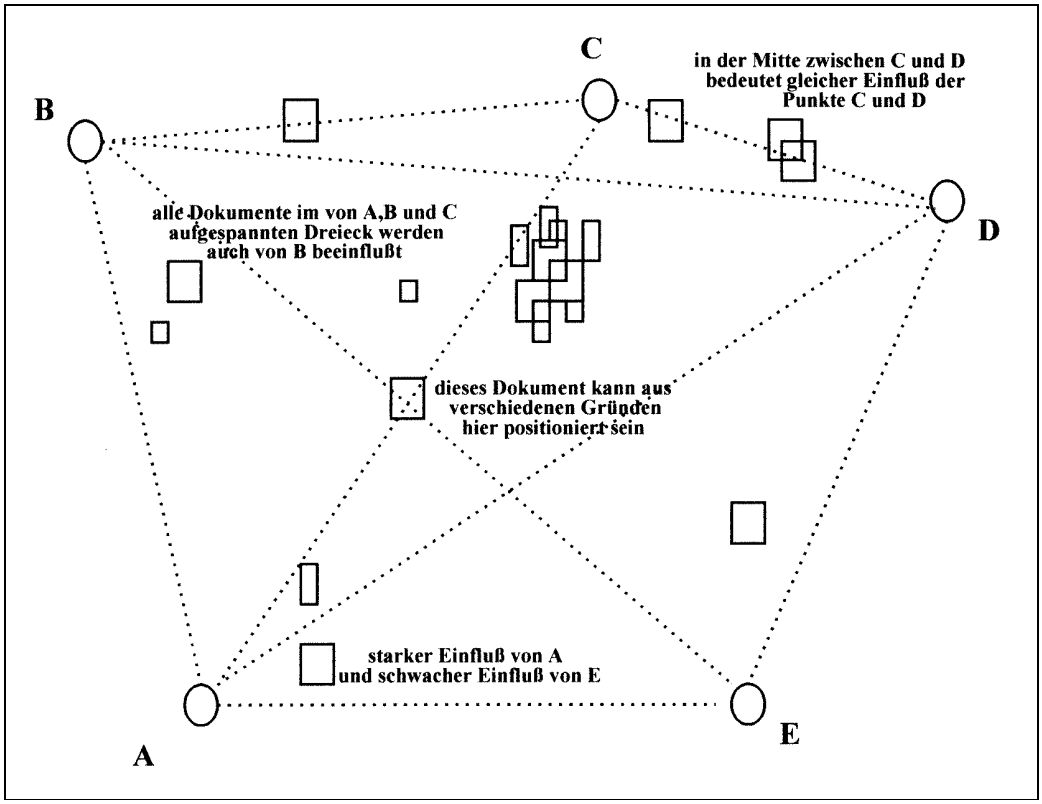

Abb. 13: VIBE-Diagramm ([Hemmje 99], S.35)

Als drittes Beispiel-Visualisierungsschema sei das VIBE-System[1] nach Olsen/ Korfhage genannt, das hier insofern von besonderer Bedeutung ist, als daß das Relevanzkugelmodell aus ihm explizit seine Inspiration bezieht ([Leissler 97], S.69f). Anstelle von Vektoren wie bei GUIDO werden hier einzelne Referenz*punkte* definiert, die die Eckpunkte eines konvexen Polygons in der Ebene bilden. Die Ergebnismenge wird in Form von Symbolen im Inneren der eingeschlossenen Fläche dargestellt, wobei sich die Position jedes einzelnen Symbols aus den Relevanzwerten der einzelnen Referenzpunkte für das betreffende Objekt ergibt: je mehr Term A für Objekt x relevant ist, um so näher wird das entsprechende Symbol am Referenzpunkt für A positioniert. Im Gegensatz zum GUIDO-Modell eignet sich diese Art der Darstellung offensichtlich auch für größere, wenn auch nicht unbedingt beliebig große, Mengen von Referenztermen; auch sie bleibt allerdings erst einmal

[1] Aus den Quellen geht nicht klar hervor, woraus sich die Abkürzung genau herleitet: Laut [Hemmje 99] ist VIBE die Kurzform von *VIsualization By Example*, nach [Chen 99] dagegen steht der Begriff für *Visual Information Browsing Environment*.

statisch, obwohl es vom Konzept her nicht schwer ist, dem Betrachter eine Möglichkeit zum Verschieben der einzelnen Referenzpunkte zu bieten.

Das zentrale Problem des VIBE-Modells besteht darin, daß die Abstände der Objekte der Ergebnismenge nicht den absoluten Relevanzwerten, sondern den Verhältnissen des jeweiligen Objekts zu allen Referenzpunkten entsprechen. Dadurch können Objekte, die eigentlich nicht viel miteinander gemeinsam haben, anders als etwa bei GUIDO im entstehenden Diagramm nahe beieinander liegen, was der intuitiven Verständlichkeit der Darstellung abträglich ist.

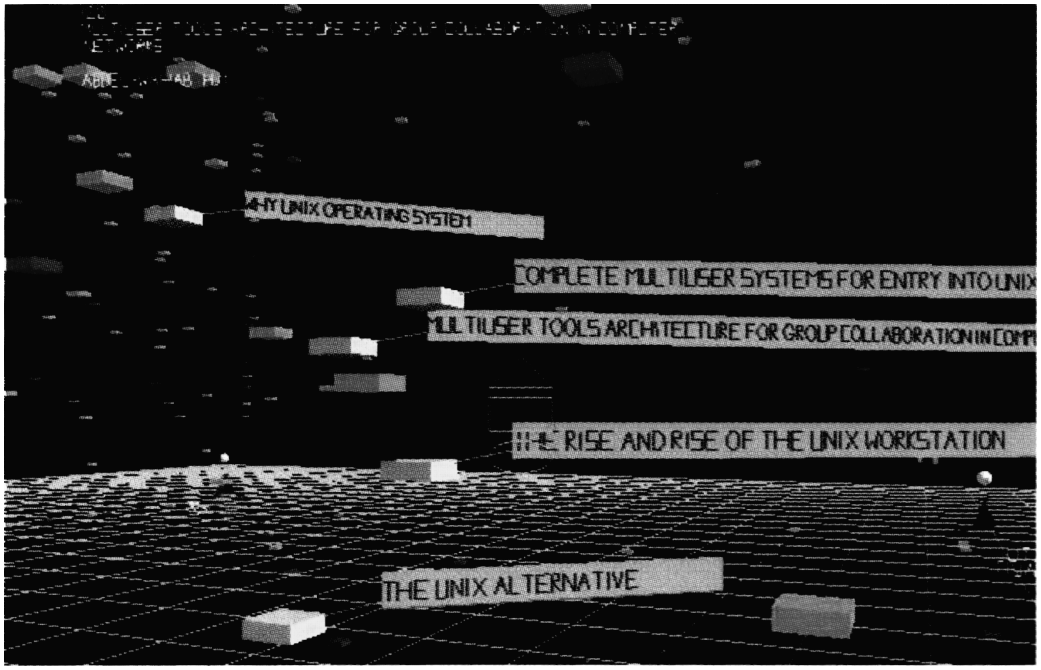

Abb. 14: VR-VIBE-Beispielszene ([Chen 99], S.108 und 190)

Es existiert heute neben dem klassischen Modell eine als VR-VIBE (alternative Schreibweise VIBE-VR) möglicherweise bekanntere Erweiterung. Diese positioniert die Referenzpunkte als Symbole in einer *dreidimensionalen* virtuellen Umgebung, zeigt grundsätzlich nur Dokumente mit einer gewissen Mindestrelevanz an, und erlaubt Bewegungen durch die Darstellung für mehrere Benutzer gleichzeitig; das grundlegende Prinzip bleibt erhalten.

2.5.4. Tree-Maps

Die sogenannten Tree-Maps fallen im Rahmen dieser Aufzählung ein wenig aus dem Rahmen, da sie zur Darstellung nicht auf die explizite Positionierung von

Objekten in einem wie auch immer definierten Raum zurückgreifen. Statt dessen handelt es sich bei ihnen primär um eine Methode zur platzsparenden Darstellung von *Baumstrukturen*. Bei der Erstellung einer Tree-Map wird, ausgehend von der Wurzel des Baums, die zur Darstellung verfügbare Fläche zwischen deren Kindern in kleinere Einzelflächen aufgeteilt, wobei sich die relativen Proportionen aus mit den Kinderknoten assoziierten Gewichten ergeben. Danach wird der Algorithmus rekursiv auf jede Teilfläche und deren zugeordneten Kinderknoten angewandt, bis der gesamte Baum traversiert und die Fläche vollständig aufgeteilt ist. Dabei können konzeptionell zusammenhängende Teilflächen etwa durch geeignete Farbkodierung von anderen abgesetzt werden ([Card 99], S. 152-159).

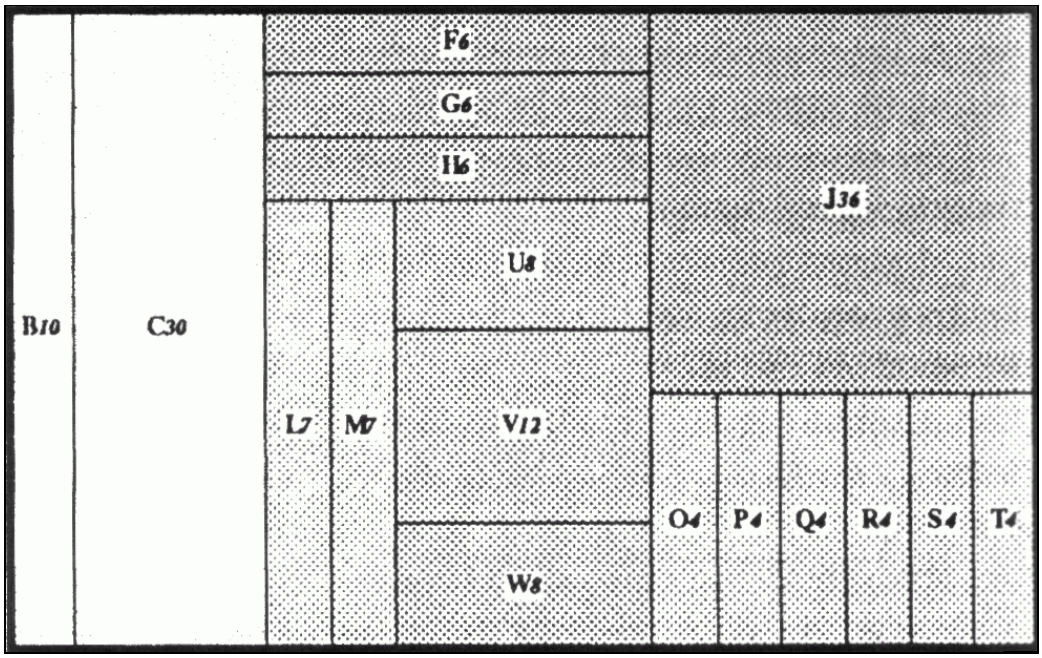

Abb. 15: Tree-Map ([Card 99], S. 154)

Wie die anderen hier beschriebenen Visualisierungsansätze eignen sich Tree-Maps prinzipiell zur Erlangung eines schnellen Überblicks über eine Gesamtinformationsmenge. Allerdings erfordert diese Art der Darstellung ein gehöriges Maß an Übung zur Interpretation, da sie definitiv kaum etwas mit der menschlichen Alltagswahrnehmungserfahrung verbindet; außerdem eignet sie sich per definitionem nur zur Darstellung baumartiger Strukturen, die nicht in jedem Fall automatisch als vorhanden angenommen werden können und eventuell erst einmal geeignet konstruiert werden müßten.

2.5.5. Narcissus

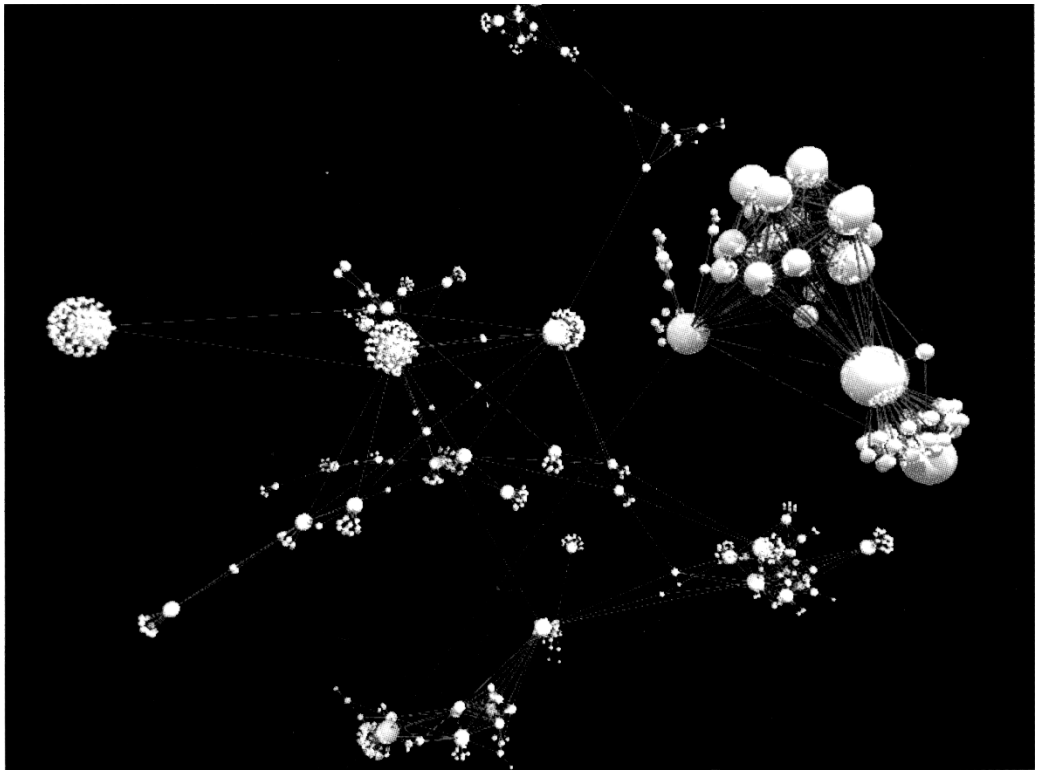

Abb. 16: Narcissus-Beispielszene ([Chen 99], S. 86)

Das 1995 in Birmingham zur Visualisierung komplexer Informationsräume entwickelte Narcissus-System (Originalartikel enthalten in [Card 99], S.503-511) verwendet ähnlich wie Bead eine Kombination aus anziehenden und abstoßenden Kräften, um die dargestellten Objekte zu positionieren; genauer gesagt, wirken zwischen zwei beliebigen Objekten erst einmal abstoßende Kräfte, während sich die Objekte, zwischen denen eine Beziehung besteht, gleichzeitig anziehen. Im Unterschied zu Bead ist die Darstellung hier hochgradig dynamisch; der menschliche Betrachter wird in die Lage versetzt, selbst neue Objekte in die Szene einzubauen bzw. aus dieser zu entfernen, zu bestimmen, welche Beziehungen erfaßt werden sollen, und die resultierenden Veränderungen in der dargestellten Szene in Echtzeit mitzuverfolgen, während die Darstellung angepaßt wird. Objekte können wahlweise durch Linien verbunden werden, die ihre Beziehung zueinander ausdrücken, oder auch nicht. Es kann sogar ausdrücklich geschehen, daß einzelne Bestandteile der Darstellung als Resultat der auf sie einwirkenden Kräfte überhaupt nicht zum Stillstand kommen, sondern letztendlich auf stabile Weise oszillieren oder andere Bestandteile umkreisen. Normalerweise führt der verwendete Algorithmus zu einer

Darstellung ähnlich Abb. 16, in der stark in Beziehung stehende Einzelobjekte deutlich erkennbare Cluster bilden.

Die offensichtliche Stärken dieses Modells sind die Anschaulichkeit, mit der es Beziehungen zwischen Informationsobjekten ausdrückt, und seine hohe Flexibilität zur Laufzeit. Zu seinen potentiellen Nachteilen gehören wie bei anderen graphisch ähnlich komplexen Systemen der relativ hohe zur Darstellung nötige Aufwand – den zwar moderne Rechnersysteme prinzipiell leicht bewältigen können sollten, der aber eventuell seine Verwendung in Situationen, in denen ein schneller Informationsfluß beispielsweise über ein Netzwerk nicht immer gewährleistet ist, einschränkt – und die Tatsache, daß der verwendete Positionierungsalgorithmus bei Manipulationen des Betrachters unter Umständen zu unerwarteten Änderungen im Modell sowie bei großen Objektmengen und komplexen Beziehungen ganz allgemein doch zu hochgradig unübersichtlichen Darstellungen führen kann.

Kapitel 3 : Verwendete Hilfsmittel

3.1. HTML

HTML (kurz für *HyperText Markup Language*), heute die Standardsprache des World Wide Web, ist weniger eine Programmiersprache im klassischen Sinn als vielmehr eine Dokumentbeschreibungs- und -formatierungssprache und damit grundsätzlich eher mit z.B. T_EX verwandt als mit C oder Pascal. Im Gegensatz zu T_EX jedoch, welches sich primär mit dem Layout und der Struktur von Druckdokumenten befaßt, wurde HTML als möglichst universeller und plattformübergreifender Standard (mit über die Möglichkeiten von einfachen Textdateiformaten hinausgehender Funktionalität) für *elektronisch* gespeicherte Dokumente entwickelt. Hierbei kommt es oft mehr auf eine brauchbare Bildschirmpräsentation unabhängig von der verwendeten Hard- und Software als auf ein möglichst präzises Drucklayout an; im Extremfall (beispielsweise für sehbehinderte Anwender) sollte es außerdem möglich sein, auf andere Ausgabemethoden zurückzugreifen.

Eine HTML-Datei, wie sie auf einem Server zum Download bereitsteht, ist prinzipiell zunächst nichts anderes als eine einfache ASCII-Textdatei, die neben ihrem eigentlichen Textinhalt zusätzlich sogenannte *Tags* enthält. Dabei handelt es sich um durch kleiner-als- und größer-als-Zeichen (gewissermaßen also spitze Klammern) vom restlichen Text getrennte kurze Zeichenfolgen mit im HTML-Standard festgelegter Bedeutung, die dem die Datei als Webseite darstellenden Browser als Gestaltungshinweise dienen. Die meisten Tags treten paarweise auf, das heißt, je ein Anfangstag <X> und ein Endtag </X> umschließen das Stück Text, auf das sie sich beziehen sollen und das seinerseits weitere Tags enthalten darf, so daß Verschachtelungen möglich sind.

Tags dienen verschiedenen Zwecken. So kennzeichnen einige beispielsweise den von ihnen eingeschlossenen Text als Strukturelement (wie beispielsweise Absatz, Überschrift, geordnete oder ungeordnete Liste – mit jeweils einzeln deklarierten Listenelementen –, oder Tabelle) mit definierten Eigenschaften. Das Paar <A>...</A> hingegen markiert wahlweise entweder einen Querverweis (Link), der im Text hervorzuheben ist, oder einen Ankerpunkt für einen solchen, der in der Regel unsichtbar bleibt, wobei das Anfangstag die jeweils nötigen Parameter aufnimmt; auf diese Weise läßt sich eine einfache Navigation innerhalb eines langen Dokuments ebenso realisieren wie das Hin- und Herwechseln zwischen verschie-

denen Webseiten selbst auf weit voneinander entfernten Rechnern. Andere Tags bezeichnen in das Dokument einzubindende nicht textbasierte Elemente (wie etwa <IMAGE> für Bilder oder <APPLET> für Java-Applets) und spezifizieren die URLs, unter denen der Browser diese auffinden kann, während wieder andere dazu dienen sollen, die Darstellung des Dokuments (etwa Betonung von einzelnen Textpassagen, Schriftarten, und Farben) genauer festzulegen.

Nicht jeder Browser versteht automatisch alle Tags. Im Laufe der Entwicklung haben sich bisher drei HTML-Standards etabliert, die aufeinander aufbauen (HTML 2, 3.2, und 4), wobei teilweise die neueren Standards Tags anbieten, die solche aus älteren obsolet machen; andererseits bleiben die älteren Tags meist aus Gründen der Rückwärtskompatibilität erhalten und werden von HTML-Autoren mitunter aus reiner Gewohnheit weiterverwendet. Dazu kommen noch proprietäre Erweiterungen der einzelnen Browserhersteller (die dann teilweise wieder in einen neuen Standard aufgenommen werden) ebenso wie plattformbedingte Einschränkungen, die dazu führen, daß ein Browser selbst bekannte Anweisungen mitunter kreativ interpretieren muß. Zum Beispiel ist einsichtig, daß ein rein textbasierter HTML-Browser keine Bilder direkt in eine Webseite einbinden kann und zu deren Darstellung die Hilfe eines anderen Programms braucht; etwas weniger klar mag dagegen auf den ersten Blick sein, daß ein solcher Browser auch bei der Darstellung von z.B. Überschriften und hervorgehobenem Text zu eigenen Hilfskonstruktionen greifen muß, da ihm auch die auf eher GUI-orientierten Plattformen verfügbaren Stilmittel wie unterschiedliche Schriftarten und -größen verwehrt sind. Aus all diesen Gründen gilt, daß ein Browser Tags, die er als solche erkennt, aber nicht zu verwerten weiß, straflos ignorieren darf, und daß selbst bei allen Optionen, die speziell HTML 4 bietet, die Möglichkeiten eines Designers, das endgültige Aussehen einer Webseite auf dem Browser eines beliebigen Betrachters von vornherein festzulegen, zwangsläufig beschränkt bleibt.

Im Rahmen des Relevanzkugelprojekts dient das World Wide Web als essentielle Grundlage. Gleichzeitig hat jedoch HTML als primär textorientierte Dokumentsprache keinerlei Möglichkeiten, Elemente wie Datenbankabfragen oder dreidimensionale Graphiken selbst zu handhaben. Diese Aufgaben müssen also mit geeigneteren Hilfsmitteln angegangen und die Lösungen dann über passende Tags in eine Webseite eingebunden werden. Java-Applets werden von seiten des HTML-Standards bereits voll unterstützt; zur Handhabung von VRML-Szenen benötigt man ein

geeignetes Plug-in für den verwendeten Webbrowser, welches dann in der Regel seine eigene Methode zur Einbindung (etwa über das <OBJECT>-Tag) mitbringt.

3.2. VRML

Die *Virtual Reality Modeling Language* VRML ist, ganz analog zu HTML für Text, eine Sprache zur Beschreibung von dreidimensionalen Grafikszenen. Entwickelt wurde sie primär zur Darstellung von virtuellen Welten, durch die sich der Betrachter hindurchbewegen und mit deren Bestandteilen er gegebenfalls interagieren kann; daher bieten VRML-Browser im Normalfall eine Auswahl an Bewegungs- und Sichtmodi, zwischen denen beispielsweise vermittels entsprechender Buttons einer Kontrolleiste hin- und hergewechselt werden kann. VRML-Szenendesign erfolgt auf relativ anschaulichem Niveau; die Sprache bietet dem Entwickler eine Auswahl an Standardbausteinen, die zur Erzeugung einer gewünschten Darstellung kombiniert werden können, und überläßt die technischen Details der Umsetzung dem Browserprogramm und dessen Hard- und Software-umgebung.

Die grundlegenden Bausteine jeder VRML-Szene sind die *Knoten*, aus denen sie sich zusammensetzt und die je nach Typ unterschiedliche Aufgaben übernehmen (siehe nächsten Absatz). Miteinander verknüpft werden die einzelnen Knoten dabei zum einen durch klassische Elter-Kind-Beziehungen, was sich auf der Quelltextebene so ausdrückt, daß einem geeigneten Feld des Elterknotens ein oder mehrere passende Kinderknoten zugeordnet werden. (Beispielsweise hat ein `Shape`-Knoten zur Darstellung eines sichtbaren Objekts normalerweise zwei Kinder – einen von mehreren möglichen Geometrieknoten, der die *Form* des Objekts definiert, und einen `Appearance`-Knoten zur Beschreibung seiner restlichen optischen Eigenschaften, der seinerseits in der Regel mindestens einen `Material`-Knoten zur Spezifikation von Farbe und Reflexionsverhalten der Objektoberfläche als Kind hat.) Davon unabhängig existiert eine zweite Verbindungsebene, auf der durch `ROUTE`-Anweisungen spezifizierte Signalwege Mitteilungen über erfolgte Ereignisse von einem Knoten zu einem oder mehreren anderen weiterleiten und dort im allgemeinen weitere Reaktionen auslösen, was sich bei komplexeren Szenen zu regelrechten Signalkaskaden ausweiten kann. Die Gesamtheit der an einer Darstellung beteiligten Knoten und Beziehungen bezeichnet man als deren *Szenengraph*.

Die verschiedenen VRML-Knotentypen lassen sich grob in folgende Kategorien einteilen:

- Knoten zur Beschreibung der eigentlichen Darstellungselemente und deren Eigenschaften wie z.B. Form, Größe, Farbe oder Textur. In diese Gruppe lassen sich auch Knoten zur Handhabung von weniger direkt offensichtlichen Elementen wie importierten Graphiken und Filmen, Licht- und Schallquellen, Szenenhintergrund, und Nebeleffekten einordnen.

- Gruppenknoten, die eine Menge von Kinderknoten zu einem zusammenhängenden Ganzen zusammenfassen und je nach Typ auch gleich Eigenschaften dieses Ganzen festlegen. Wahrscheinlich das am häufigsten auftretende Beispiel hierfür ist der `Transform`-Knoten, der für die von ihm abhängigen Knoten ein neues, modifiziertes lokales Koordinatensystem festlegt und auf diese Weise bei der Positionierung, Drehung, und Skalierung praktisch aller Objekte in einer Szene verantwortlich zeichnet.

- Zeit- und Sensorknoten, die ausgehende Ereignisse generieren, wenn je nach Knotentyp bestimmte Bedingungen erfüllt sind. Typisch sind etwa Knoten zum Abfangen von Mausereignissen über einem bestimmten Teil der Szenengeometrie oder zum Starten bzw. Stoppen von Abläufen zu gegebener Zeit. Verwandt mit dieser Gruppe sind außerdem die Interpolatorknoten zur Generierung von nicht explizit angegebenen Zwischenwerten anhand einer vorgegebenen Liste von Eckdaten, die typischerweise bei Animationsaufgaben Verwendung finden.

- Skriptknoten, die in einer geeigneten Skriptsprache das Verhalten der Szene oder einzelner Bestandteile derselben beschreiben. Diese Gruppe enthält tatsächlich nur einen einzigen Knoten – `Script` –, der auch der einzige Standardknoten ist, dessen Schnittstelle zum Rest der Szene nicht explizit vom Standard selbst festgelegt ist, sondern von seinem Benutzer eigens definiert wird. Der Standard beschreibt die Verwendung von Java und Javascript als mögliche Skriptsprachen ([VRML 97], Anhang B bzw. C).

Neben dem gegebenen Bestand an Standardknoten bietet VRML außerdem die Möglichkeit, eigene neue Knotenprototypen zu definieren, indem bereits vorhandene Knoten geeignet kombiniert und als Gesamtheit mit einem neuen Typnamen und Interface versehen werden.

Ein wesentlicher Vorteil von VRML gegenüber anderen 3D-Grafiksystemen liegt darin, daß die Sprache erstens nicht proprietär und zweitens durch den entsprechenden ISO-Standard ([VRML 97]) klar spezifiziert ist; eine VRML-Szene, die sich an diese Spezifikation hält und keine browsereigenen Erweiterungen verwendet, kann prinzipiell auf jedem standardkonformen VRML-Browser problemlos dargestellt werden. (Die Standardkonformität der existierenden VRML-Browser steht allerdings eventuell auf einem anderen Blatt – mehr dazu später in Abschnitt 6.1. .) Außerdem sind VRML-Plugins für gängige Webbrowser – zumindest für den allgegenwärtigen Internet Explorer von Microsoft, je nach Hersteller mitunter auch einmal für andere Modelle – oft kostenlos zum Download aus dem World Wide Web erhältlich; damit können VRML-Szenen als Objekte in normale Webseiten aufgenommen und, ein entsprechendes Plugin beim Empfänger vorausgesetzt, auch Web-weit übertragen und dargestellt werden. Eine zweite Sprache, die denselben Aufgabenbereich abdeckt und auch nur annähernd dieselbe (wenn auch begrenzte) Verbreitung gefunden hat, existiert zur Zeit nicht.

Allerdings soll an dieser Stelle nicht verschwiegen werden, daß VRML als potentielles Hilfsmittel zur Erstellung von webbasierten 3D-Graphikszenen auch einige Schwächen hat, die sich zum Teil auf das Alter des gegenwärtigen Sprachstandards von 1997 zurückführen lassen. Im Einzelnen lassen sich folgende Kritikpunkte aufzählen:

- Die Sprache ist nicht *völlig* frei erweiterbar. Neue Arten von Knoten können nur aus bereits existierenden Prototypen zusammengesetzt, aber nicht von Grund auf eigenständig definiert werden, und eine Erweiterung der vorgegebenen Auswahl an Datentypen für Felder und Signale ist überhaupt nicht vorgesehen.

- Darzustellende Objekte werden ausschließlich über ihre Oberflächen definiert. Das bedeutet, daß eher volumenbezogene 3D-Modellierungsansätze nicht zur Anwendung kommen können bzw. gegebenfalls unter Verwendung geeigneter Skriptknoten erst einmal entsprechend übersetzt werden müßten.

- Überhaupt lassen sich komplexe Objekte nur durch rein additives Zusammensetzen einfacherer Teilformen oder, bei Verwendung einiger spezielle Knoten wie `IndexedFaceSet` oder `Extrusion`, durch Polygonnetze mit explizit definierten Eckpunkten und Kanten modellieren. Eine Unterstützung für die

Darstellung echter gekrümmte Flächen fehlt im allgemeinen[1] ebenso wie andere Methoden, Einzelbestandteile zu kombinieren. (So ist es in VRML einfach, einen Würfel darzustellen, der von einem Zylinder durchbohrt wird. Will man aber den Zylinder statt dessen vom Würfel quasi abziehen, um ein Loch mit dem gleichen Durchmesser zu erhalten, so zeigt sich schnell, daß diese Möglichkeit unter VRML einfach nicht *existiert*.)

- Interaktionen zwischen einzelnen dargestellten Objekten werden kaum unterstützt. Zum Beispiel modelliert VRML keine indirekte Beleuchtung (das Aussehen von Objekten wird nur durch ihre Oberflächeneigenschaften und die vorgegebenen Lichtquellen definiert, nicht aber durch theoretisch von anderen Objekten reflektiertes Licht beeinflußt), Objekte werfen keine Schatten, und es existieren keine Sensorknoten zur Erkennung von Annäherungs- und Kollisionsereignissen zwischen beliebigen Objekten (die vorhandenen `Collision`- und `ProximitySensor`-Knoten reagieren ausschließlich auf den virtuellen Avatar des Beobachters selbst).

- Gerade weil VRML auf relativ hohem Level operiert, sind gegebenenfalls Informationen über implementierungsnahe Details der resultierenden Darstellung – wie beispielsweise die aktuellen Dimensionen eines abgebildeten Objekts *auf dem Bildschirm selbst* – nicht verfügbar und können damit auch nicht verwendet werden, um die Szene bei möglicherweise entstehendem Bedarf anzupassen.

Die meisten dieser Schwächen beeinträchtigen die Eignung der Sprache zur Implementierung eines Relevanzkugelwerkzeugs nicht wesentlich, da die einzelnen Bestandteile des Modells zunächst einmal nicht übermäßig komplex sind. Einzig die Abwesenheit einer Möglichkeit, Zusammenstöße zwischen Objekten zu erkennen und darauf zu reagieren oder zumindest die aktuellen 'physischen' Dimensionen eines Objekts abzufragen, macht sich im Zusammenhang mit dem Problem der unbeabsichtigten Überlappung von Datenobjekten störend bemerkbar.

[1] Es existiert eine Erweiterung des Standards – [VRML A 02] –, die eine optionale Darstellung von gekrümmten Linien und Flächen mit Hilfe von NURBS *(Non-Uniform Rational B-Splines)* vorsieht und zu diesem Zweck nicht weniger als zehn neue Knotenprototypen einführt. VRML-Browser müssen diese Erweiterung jedoch nicht unterstützen.

3.3. Java

Ab 1991 im Rahmen des Green-Projekts entstanden und 1995 der Öffentlichkeit vorgestellt, ist Java heute eine der verbreiteteren Programmiersprachen. Bekannt wurde die Sprache zunächst als Mittel zur Realisierung meist kleiner lauffähiger Programme, die direkt in Webseiten eingebunden werden und dort mehr oder weniger nützliche Aufgaben übernehmen konnten – die klassischen Java-*Applets*. Im Gegensatz zu einfachen Skriptsprachen (wie dem etwas mißverständlich benannten JavaScript, das tatsächlich mit Java wenig gemein hat und an dessen Leistungsfähigkeit nicht annähernd heranreicht) ist Java jedoch eine vollwertige objektorientierte Programmiersprache, deren Wortschatz teilweise sichtbar von C und C++ entlehnt ist, die aber auf einige von deren komplexeren und fehleranfälligen Aspekten wie beispielsweise Zeigerarithmetik und explizite Anforderung und Freigabe von Speicher durch den Programmcode zur Laufzeit verzichtet.

Charakteristisch für Java sind folgende Merkmale:

- Java ist sehr stark objektorientiert; abgesehen von wenigen grundlegenden primitiven Datentypen und einem kleinen Grundwortschatz von Befehlen besteht im Prinzip die gesamte Sprache aus Klassenbibliotheken für die verschiedensten Arten von Objekten. So ist zum Beispiel ein Java-Programm selbst nichts anderes als ein Objekt, das zu einer vom Programmierer neu definierten Klasse gehört, die bestimmte feste Standardfunktionen (bzw. im Java-Sprachgebrauch Standard*methoden*) implementieren muß, um das letztendlich resultierende Programmobjekt lauffähig und aufrufbar zu machen. Dabei erlaubt Java natürlich sowohl die Wiederverwendung bereits vorhandener Objektklassen an sich als auch deren Erweiterung durch entsprechende Vererbungsmechanismen.

- Java ist weitgehend plattformunabhängig. Im Gegensatz zu Sprachen wie C, Fortran oder Pascal erzeugt ein Java-Compiler keine Folge von Befehlen einer prozessor- und systemabhängigen Maschinensprache, sondern einen speziellen und auf die sogenannte virtuelle Java-Maschine zugeschnittenen Bytecode. Dieser Code ist erstens kompakt und kann zweitens auf jedem System ausgeführt werden, das in der Lage ist, diese virtuelle Architektur zu emulieren; beispielsweise sind virtuelle Java-Maschinen Bestandteil jedes Web-Browsers mit funktionierender Java-Unterstützung.

- Java-Applets im Besonderen sind (in einer perfekten Welt) normalerweise *sicher.* Da ein Applet für gewöhnlich auf einem anderen Rechner ausgeführt wird als dem, der die entsprechende Webseite zum Herunterladen zur Verfügung stellt, haben die Entwickler der Sprache von vornherein dafür Sorge getragen, daß Applets auf dem fremden System nur im Rahmen des Erlaubten tätig werden können und ansonsten isoliert sind. (Das typische Bild, das hier verwendet wird, ist das vom Applet-eigenen 'Sandkasten', innerhalb dessen es problemlos 'spielen', den es aber nicht verlassen kann.) Beispielsweise haben Applets ohne explizite Erlaubnis des Benutzers keinen Zugriff auf das Dateisystem des Rechners, auf dem sie laufen, können also weder Dateien ausspionieren noch löschen oder auch nur eigene anlegen[1].

Der VRML-Standard sieht (in erster Linie wohl wegen der bereits vorhandenen Anbindung an das World Wide Web und damit an die Browserprogramme, auf denen gängige VRML-Browser oft aufsetzen, ebenso wie der Kompaktheit und relativen Systemunabhängigkeit des Java-Bytecodes) die Möglichkeit vor, das Verhalten von Skriptknoten unter Verwendung entsprechender Java-Klassen und -Methoden zu definieren. Außerdem gehören zum Java-Sprachumfang Bibliotheken mit Klassen zum Zugriff sowohl auf Netzfunktionen im allgemeinen als auch speziell auf SQL-taugliche Datenbanken; daneben ist Java eine von zur Zeit nur zwei Sprachen, deren Unterstützung im ISO-Standard zum External Authoring Interface, mit dessen Hilfe externe Programme auf VRML-Szenen zugreifen können (siehe auch nächsten Abschnitt), explizit vorgesehen ist. All das macht die Sprache zu einem natürlichen Kandidaten für die Implementierung der *Funktionalität* unseres VRML-basierten Relevanzkugelmodells, ebenso wie potentiell für Applets und Programme, die dieses Modell dann als Werkzeug einsetzen.

3.4. Das External Authoring Interface (EAI)

Aus einer Sammlung von Java-Klassen zur Ansteuerung einer VRML-Szene durch ein auf der gleichen Webseite befindliches Applet ([Marrin 97]) hat sich das sogenannte *External Authoring Interface* zu einem eigenständigen Teil der VRML-Spezifikation gemausert ([EAI]), auch wenn es strenggenommen nicht Bestandteil der Sprache VRML selbst ist. Statt dessen spezifiziert das letztere Dokument eine

[1] Tatsächlich ist allein schon die *Möglichkeit*, einem Applet zu erlauben, diese Sicherheitsbestimmungen zu umgehen, erst seit Java-Version 1.2 Bestandteil der Sprache.

prinzipiell sprachunabhängige Schnittstelle, über die potentiell beliebige Anwendungen von außerhalb auf eine VRML-Szene zugreifen, Informationen abrufen, und gegebenenfalls Anweisungen erteilen können. [EAI] beschreibt in den Anhängen A und B beispielhaft die Einbindung der Schnittstelle unter IDL und Java.

Die EAI-Schnittstelle erlaubt externen Programmen den Zugriff sowohl auf den VRML-Browser und damit globale Informationen und den Szenengraphen an sich, als auch auf einzelne Knoten, soweit diese eindeutig identifiziert werden können – beispielsweise über einen szeneninternen eigens vergebenen Namen, aber auch einfach über geeignete Zeiger, soweit das externe Programm selbst für die Erzeugung der Knoten verantwortlich zeichnet – und deren Signalein- und -ausgänge. Dabei kann das externe Programm ebenso Ereignisse an einzelne Knoten in der Szene schicken wie von diesen ausgehende Ereignisse nach vorheriger expliziter Anknüpfung an die entsprechenden Ausgänge selbsttätig empfangen und verarbeiten. In ihrer modernen Fassung kann die Schnittstelle externe Zugriffe von verschiedensten Seiten auf einen VRML-Browser handhaben, der seinerseits durchaus auch auf einem entfernten Rechner laufen kann.

Die in [Marrin 97] beschriebenen Java-Klassen sind heute strenggenommen obsolet, zumal sie nur den Spezialfall der Interaktion zwischen VRML-Szene und Applet auf ein und derselben Seite behandeln, während die modernere Standardfassung potentiell alle Formen des externen Zugriffs auf VRML-Browser abdeckt. Allerdings ist ihre Beschreibung im angegebenen Dokument leichter einsichtig und stellenweise detaillierter als die ihrer Gegenstücke in [EAI], und aus Gründen der Rückwärtskompatibilität werden sie von derzeitigen Browsern durchaus noch unterstützt.

Offensichtlich ist damit das External Authoring Interface *das* potentielle Bindeglied zwischen einer VRML-Relevanzkugel und irgendwelchen Anwendungen, die auf sie zugreifen wollen. Es existieren ohnehin keine bekannten alternativen Ansätze zur externen Ansteuerung einer VRML-Szene, und die Frage der Definition einer relevanzkugelspezifischen Schnittstelle reduziert sich bei einer Verwendung des EAI praktisch auf die Spezifikation der einzelnen Datenfelder sowie Signalein- und -ausgänge des Relevanzkugelknotens selbst.

Kapitel 4 : Überlegungen zum Entwurf

4.1. Anforderungsanalyse

Ziel dieser Arbeit ist eine möglichst hardware- und betriebssystemunabhängige Implementierung des Relevanzkugelmodells auf Grundlage des World Wide Web unter Verwendung von VRML und Java. Dabei soll sowohl dem Betrachter als auch externen Anwendungen, die an das Relevanzkugelwerkzeug ankoppeln, eine einfache und intuitive Benutzung ermöglicht werden.

Um den an sie gestellten Ansprüchen zu genügen, muß die neue Implementierung der Relevanzkugel sich einer Reihe von Einzelanforderungen stellen:

1.) Sie soll natürlich zunächst einmal das Modell komplett abbilden. Dabei sollte gleichzeitig nach Möglichkeiten gesucht werden, dessen in 2.4. angesprochene Schwächen zu kompensieren oder zumindest deren Auswirkungen abzumildern. Wenn sich daneben allgemein Anschaulichkeit und/ oder Informationsgehalt der Darstellung verbessern lassen, ohne gleichzeitig die Verwendbarkeit des Modells durch zu starke Spezialisierung einzuschränken, sollte die Gelegenheit dazu ebenfalls genutzt werden.

2.) Ihre Bedienung (beziehungsweise Ansteuerung) soll sowohl für den menschlichen Benutzer als auch für auf derselben Seite eingebettete Java-Applets möglichst einfach sein.

3.) Sie muß unabhängig von der eventuellen Komplexität der endgültigen Implementierung ihre Fähigkeit behalten, auch bei möglichst großen darzustellenden Objektmengen noch in wahrnehmbarer Echtzeit auf die Eingaben des Benutzers zu reagieren. (Dies mag in Anbetracht der Leistungsfähigkeit moderner Systeme etwas übervorsichtig klingen. Jedoch wird Java-Code normalerweise um ein Mehrfaches langsamer abgearbeitet als systemeigene Maschinensprachen, da die Java-Maschine erst emuliert werden muß; außerdem laufen auf denselben modernen Systemen unter gängigen Betriebssystemen in der Regel noch eine Vielzahl von anderen Prozessen, mit denen sich der VRML-Browser, die einzelnen Knotenskripte, und die externe Anwendung, die die Relevanzkugel verwendet, die Systemressourcen teilen müssen.)

4.) Sie soll auf möglichst vielen Plattformen lauffähig sein, muß also existierenden Standards (sowohl offiziellen wie der VRML-Standarddefinition als auch eher inoffiziellen de-facto-Standards) entsprechen.

5.) Wenn möglich, soll der endgültige Code nicht zu umfangreich sein, um einen eventuellen Download von einer Webseite auch für Verbindungen mit begrenzter Bandbreite nicht unnötig in die Länge zu ziehen.

4.2. Teilaufgaben

Grundsätzlich kann man sich die Implementierung des Relevanzkugelwerkzeugs unabhängig von der verwendeten Plattform aus folgenden einzelnen Teilaufgaben zusammengesetzt denken:

1.) Implementierung der Referenzobjekte;

2.) Implementierung der einzelnen Datenobjekte;

3.) gleichzeitig und komponentenübergreifend die Handhabung der Interaktionen zwischen den einzelnen Objekten; und letztlich

4.) Implementierung einer Schnittstelle nach 'außen'.

Da VRML es gestattet, eigene neue Knotentypen aus bereits vorhandenen zusammenzustellen und gegebenenfalls deren Verhalten durch Verwendung geeignet definierter Skriptknoten festzulegen, liegt die Idee nahe, jeweils einen spezialisierten Prototypen für Referenz- bzw. Datenobjekte zu definieren und diese über geeignete Signalwege so miteinander kommunizieren zu lassen, daß sich die gewünschte Gesamtfunktionalität ergibt. Ein dritter Prototyp kann definiert werden, um die Relevanzkugel als Ganzes zu erfassen und quasi einzukapseln; auf diese Weise sind die einzelnen Komponenten vor Zugriffen von außen, die ihr Zusammenspiel störend beeinflussen könnten, einigermaßen geschützt[1], und es bleibt gleichzeitig der zugreifenden Partei, ob nun menschlicher Programmierer oder externes Programm, einige Verwaltungsarbeit wie etwa das explizite Festlegen der einzelnen Signalrouten erspart, da diese kugelintern automatisch erfolgen kann. Die Reali-

[1] Man denke sich etwa den Fall, daß durch Signale von einer anderen Anwendung den einzelnen Referenzobjekten *unterschiedliche* Abstände vom Kugelzentrum zugewiesen würden.

sierung der Schnittstelle für externe Anwendungen kann über die Standardzugriffsfunktionen des External Authoring Interface erfolgen, indem man einfach auf der Interfacespezifikation für den Relevanzkugel-Knotenprototyp aufsetzt. Idealerweise sollten alle diese Komponenten möglichst unabhängig voneinander sein, um spätere Verbesserungen und Erweiterungen der Implementierung zu erleichtern.

An spätestens diesem Punkt stellt sich die Frage nach der Anzahl der Parameter, die nötig sind, um eine Relevanzkugel vollständig zu beschreiben. Offensichtlich gehört dazu auf jeden Fall der Kugeldurchmesser bzw. -radius, der den Abstand aller Referenzobjekte von der Kugelmitte gleichermaßen festlegt. Ferner bietet es sich an, die *Positionen* der Referenzobjekte in einer Form von Polarkoordinaten auszudrücken; nachdem der Abstand vom Kugelzentrum für alle gleich ist, reduziert sich diese Aufgabe auf die Angabe von Drehwinkel und -richtung relativ zu einer fest vorgegebenen Ausgangsposition, etwa durch Standard-VRML-Rotationswerte. Hat man jetzt noch die Gewichte der einzelnen Referenzobjekte und eine Möglichkeit, sie eindeutig zu identifizieren, so sind sie *prinzipiell* vollständig beschrieben; dazu kommen natürlich noch Faktoren wie Größe, Form, und Farbgebung zur eigentlichen Darstellung sowie gegebenenfalls zur Vermittlung von Informationen und Verdeutlichung von Zusammenhängen.

Für eine vollständige Bestimmung der *Daten*objekte genügen bei genauerer Betrachtung bei einmal gegebenen Referenzobjekten (mit bekannten Positionen und Gewichten) wieder eine eindeutige Unterscheidungsmöglichkeit der einzelnen Objekte und pro Datenobjekt eine Liste von Relevanzfaktoren, die auf die bekannten Referenzobjekte Bezug nehmen. Damit ist bei fester gegebener Rechenvorschrift die Position jedes einzelnen Datenobjekts im kugeleigenen Koordinatensystem prinzipiell bereits eindeutig festgelegt und muß nicht notwendigerweise noch einmal explizit festgehalten werden. Auch hier kommen in der Darstellung noch Form-, Größen-, und Farbaspekte hinzu. Da gerade die Anzahl der Datenobjekte je nach Anwendung recht groß werden kann, kann es sich einerseits empfehlen, diese möglichst einfach zu gestalten, um es dem Betrachter zu erleichtern, den Überblick über die Darstellung zu behalten; andererseits ließe sich potentiell gerade über diese Faktoren wiederum deren Informationsgehalt erhöhen.

Zu all den angesprochenen Parametern kommt noch die Forderung nach einer Schnittstelle, die einerseits alle wichtigen und andererseits möglichst wenig überflüssige Informationen sowohl von der Relevanzkugel an andere Anwendungen als

auch umgekehrt weiterleiten kann. Dazu kann beispielsweise gehören, mit welchem Objekt der Betrachter gerade in welcher Weise interagiert; eher interne Details der *Darstellung* wie beispielsweise die konkret ermittelten einzelnen Objektpositionen bleiben dagegen, soweit sie nicht bewußt von außen veränderbar sein sollen, besser innerhalb des Kugelknotens eingeschlossen.

4.3. Referenzobjekte

Obwohl sie rein optisch komplexer wirken mögen als die in den bisherigen Versionen verwendeten Datenobjekte, sind die Referenzobjekte doch in erster Linie einfache Bedienungselemente, deren Aufgabe darin besteht, zu warten, bis der Betrachter sie beispielsweise durch Erfassen mit dem Mauszeiger manipuliert, daraufhin ihre Position und/oder ihren Gewichtungsfaktor zu ändern, und diese Änderung an die einzelnen Datenobjekte, für die sie relevant sind, weiterzuleiten, so daß diese *ihre* Positionen entsprechend anpassen können. Ein derartiges Verhalten ist mit Standardknoten aus dem VRML-Sprachschatz und einem geeigneten Skript relativ einfach zu realisieren.

In der Darstellung sitzen die Referenzobjekte normalerweise auf der Oberfläche der Relevanzkugel auf[1]. In Bezug auf ihre Gestaltung ist die bei der in [Leissler 97] vorgestellten Version verwendete Form schon nahezu optimal: eine Kugel, die frei über die Oberfläche der Relevanzkugel verschoben werden kann und deren Größe einen groben Anhaltspunkt für das aktuelle Gewicht liefert, ausgestattet mit einem Knopf, der Änderungen des Gewichts durch Hineindrücken in bzw. Herausziehen aus dem Kugelteil erlaubt, und einem freischwebenden Schriftzug zur eindeutigen Identifizierung des Objekts. Die einzige wirkliche Schwäche dieser Form besteht darin, daß das Referenzobjektgewicht eben *nur* implizit über die Objektgröße angedeutet wird, zumal diese im dreidimensionalen Modell zwangsläufig auch mit der Entfernung des Objekts zum Betrachter variiert. Dies läßt sich jedoch einfach beheben, indem man beispielsweise dem bereits vorhandenen Schriftzug eine explizite numerische Gewichtsangabe beifügt.

Es stellt sich noch die Frage nach einer geeigneten Anfangsverteilung der Referenzobjekte auf der Kugelschale. Natürlich soll eine Möglichkeit geboten werden, ihre

[1] Eventuell kann – aus eher kosmetischen Gründen – ein optionaler Zusatzabstand hinzukommen, der Teilkollisionen zwischen Referenz- und Datenobjekten (die ja ihrerseits auch eine gewisse Größe haben) verhindern helfen soll.

Startpositionen auf Wunsch ausdrücklich festzulegen; ebenso natürlich wird aber auch eine sinnvolle, dem Betrachter möglichst sofort einleuchtende Voreinstellung gebraucht. Hier bieten sich zwei mögliche Ansätze:

- Die Referenzobjekte werden gleichmäßig so über die Kugelschale verteilt, daß sie die Eckpunkte eines regelmäßigen, beispielsweise platonischen, Körpers bilden. Auf diese Weise werden sowohl der verfügbare Platz auf der Kugeloberfläche als auch der Kugelinnenraum optimal genutzt; allerdings kann es geschehen, daß ein oder mehrere Referenzobjekte zunächst schlecht sicht- und erreichbar *hinter* der Relevanzkugel und einer potentiell dichten Wolke von Datenobjekten in deren Innerem positioniert werden. Daneben stellt sich auf der Implementierungsseite das Problem, eine Methode zu finden, die diese Art von Verteilung für eine beliebige Anzahl von Referenzobjekten zufriedenstellend berechnet.

- Die Referenzobjekte bilden einen Ring, idealerweise in der Bildschirmebene. Dieser Ansatz ist einfacher zu realisieren und bietet dem Betrachter sofortigen Zugriff auf jedes einzelne Referenzobjekt, nutzt aber natürlich die dritte Dimension der Darstellung nicht aus, da alle Referenz- und Datenobjekte in diesem Fall zunächst in einer Ebene liegen. Auch lassen sich auf diese Weise potentiell nicht so viele Referenzobjekte positionieren wie bei der ersten Methode, bevor es zu gegenseitigen Überlappungen kommt; dieser Nachteil relativiert sich allerdings, da immer noch die Möglichkeit besteht, die Objekte nach dem Start der Relevanzkugel neu zu ordnen und sich im allgemeinen zumindest bei der Repräsentation *menschlicher* Informationsanfragen die Anzahl der Suchbegriffe und damit der Referenzobjekte in überschaubaren Grenzen halten sollte.

4.4. Datenobjekte

Die im Inneren der Relevanzkugel befindlichen Datenobjekte symbolisieren in der Regel einzelne Dokumente, Datensätze, oder ähnliche Bestandteile einer größeren Menge von potentiell interessanten Informationen. Ihre Positionen ergeben sich aus der Gewichts- und Positionsverteilung der einzelnen Referenzobjekte unter Einbeziehung der aktuell gültigen Relevanzfaktoren, die normalerweise für eine bestimmte Kombination von Daten- und Referenzobjekten konstant sind, und die primäre Interaktionsmöglichkeit mit ihnen, die das Modell dem Benutzer bietet, besteht darin, einzelne Datenobjekte z.B. vermittels Mausklick anzuwählen und zur

besseren Hervorhebung aus der Masse zu markieren, bzw. die Markierung in analoger Weise wieder aufzuheben. All dies klingt zunächst einmal recht einfach, sobald man erst einmal eine geeignete Rechenvorschrift zur Positionsbestimmung gefunden hat; tatsächlich jedoch wirft das Design der Datenobjekte von allen Teilaufgaben die weitaus meisten Fragen auf.

4.4.1. Form

Zunächst einmal stellt sich die scheinbar banale Frage nach einer geeigneten *Form* – wobei wir hierunter der Einfachheit halber auch Aspekte wie Größe, Farbe und dergleichen zusammenfassen wollen. Die erste LyberWorld-Relevanzkugel verwendete eigens ausmodellierte Buchsymbole, da die damalige Anwendung auf der Basis von Textdokumenten arbeitete; in [Leissler 97] wurden diese durch einfacher modellierbare und für generische Anwendungen potentiell sinnvollere Würfel bzw. Quader ersetzt. Keine der beiden Varianten leistet in der Praxis mehr als die reine Markierung einer Position im Kugelinnenraum. Das ist an sich nichts Verwerfliches, zumal sich die Aussagekraft des Modells ja in erster Linie auf die Verteilung der einzelnen Objekte im Raum stützt. Will man jedoch innerhalb der Relevanzkugeldarstellung zusätzliche Informationen vermitteln, etwa zur genaueren Identifizierung einzelner Datenobjekte oder zur besseren Verdeutlichung der Zusammenhänge von Referenz- und Datenobjekten, so kann es sich lohnen, über Alternativen nachzudenken.

4.4.2. Identifizierung

Zur Identifizierung einzelner Datenobjekte bietet sich, wenn man diese nicht einfach entsprechenden Möglichkeiten der externen Anwendung überlassen will, die die Daten liefert, noch am ehesten die Einblendung eines kurzen kennzeichnenden Schriftzugs oder anderen Labels in der Nähe des jeweiligen Objekts an – ganz analog zu derselben Praxis bei den Referenzobjekten. Allerdings wird in vielen Anwendungsfällen die *Anzahl* der Datenobjekte die der Referenzobjekte deutlich, durchaus schon einmal um Größenordnungen, übersteigen, und da sie außerdem im Kugelinneren zusammengepackt sind, stehen sie oft wesentlich dichter beieinander. Eine Kennzeichnung jedes einzelnen Datenobjekts in der Darstellung verbietet sich also schon aus Gründen der Übersichtlichkeit. Dagegen steht eine vorübergehende Einblendung für ein Datenobjekt, welches gerade vom Mauszeiger berührt wird, möglicherweise kombiniert mit einer persistenten Kennzeichnung der gerade

markierten Objekte, eher unter der Kontrolle des Betrachters und klingt daher praktikabel.

4.4.3. Assoziation mit Referenzobjekten

Zur Hervorhebung der konzeptionellen Verbindung zwischen Referenz- und Datenobjekten gibt es mindestens drei verschiedene Möglichkeiten: explizite Verbindungslinien, geeignetes Hervorheben der assoziierten Objekte der jeweils anderen Kategorie, und/oder geeignete Änderungen der Form oder Ausrichtung der Datenobjekte, um die von den Referenzobjekten ausgehenden hypothetischen Anziehungskräfte zu symbolisieren.

Der erste Ansatz hat den potentiellen Vorteil, daß seine Bedeutung leicht erfaßbar ist und die Linien dem Auge des Betrachters als zusätzliche Orientierungshilfen dienen könnten; allerdings eignet sich diese Lösung aus Gründen der Übersichtlichkeit nur für einzelne beziehungsweise wenige ausgewählte Datenobjekte zu jedem gegebenen Zeitpunkt, da *zuviele* Linien wiederum den Blick des Betrachters auf andere Darstellungselemente behindern würden.

Die zweite Methode – Hervorhebung assoziierter Objekte, im Prinzip ähnlich, wie sie für ausgewählte Datenobjekte ohnehin schon vorgesehen ist – bewahrt diese Übersichtlichkeit, bietet aber keine ganz so klare Assoziation und kann insbesondere dann, wenn einzelne Objekte momentan nicht im darstellenden Fenster selbst enthalten sind, dazu führen, daß der Betrachter tatsächlich existierende Zusammenhänge in der Praxis übersieht.

Der dritte Ansatz bestünde nun darin, die Form der Datenobjekte selbst komplexer, eventuell sogar völlig flexibel zu gestalten, so daß sich die auf das einzelne Datenobjekt einwirkenden Kräfte, die konzeptionell von den relevanten Referenzobjekten ausgehen, in Echtzeit durch entsprechende Formveränderungen ausdrükken könnten. (Prinzipiell könnten sich die Datenobjekte auch etwa in der Art einer Kompaßnadel nach diesen Kräften ausrichten – es ist allerdings nicht klar, inwieweit dieser Ansatz bei aus *mehreren* Richtungen simultan angreifenden Kräften die nötige Aussagekraft liefern könnte.) Die Kunst der dreidimensionalen Computergraphik kennt in der Tat Ansätze, die eine entsprechende Modellierung unterstützen würden, wie beispielsweise die sogenannten *Metaballs* – Objekte, deren Oberflächenform durch eine Potentialfunktion und einen Grenzwert für diese spezifiziert wird, der angibt, welche Punkte als inner- beziehungsweise außerhalb

des Objekts liegend betrachtet werden. Da die Stärke der auf ein Datenobjekt einwirkenden Kräfte nur von Gewichts- und Relevanzfaktoren abhängt, sind sie distanzunabhängig und, soweit aktuell an keinem Referenzobjekt manipuliert wird, auch zeitlich konstant; primärer Faktor für die Verformung einer flexiblen Datenobjektoberfläche wären also die Angriffswinkel und die Überlagerung der einzelnen Funktionen.

Unglücklicherweise wird die für den dritten Ansatz erforderliche Modellierung flexibler Formen von VRML in keinster Weise direkt unterstützt. Die einzige verfügbare Methode zur Modellierung komplexerer Objekte ist – neben der Kombination von grundlegenderen Einzelkomponenten – das Zusammensetzen von Polygonnetzen aus explizit vorgegebenen Eckpunkten und verbindenden Kanten. Das heißt, daß beim Versuch einer Umsetzung dieser Idee die gesamte zur Modellierung der Deformationen nötige Rechenarbeit von einem geeigneten Skript selbst geleistet werden müßte. Gleichzeitig müßte das Polygonnetz feinmaschig genug sein, um ein gewisses Mindestmaß an Detail darstellen zu können; ideal wäre wahrscheinlich eine angenäherte Kugelform mit genügend einzelnen Punkten, um auch noch Ausbuchtungen, die bereits an der Basis miteinander zu verschmelzen beginnen, bis zu einer gewissen Grenze an ihren höchsten Punkten unterscheiden zu können. All dies führt potentiell zu einer sehr rechenaufwendigen Implementierung, insbesondere im Vergleich mit den in bisherigen Versionen verwendeten einfachen soliden Formen, die zum einen von vornherein mit deutlich weniger Eckpunkten, Kanten, und Flächen auskommen und deren Eckpunkte sich zum anderen bei Bedarf aus ihrer Position schon durch einfache Addition von Konstanten ergeben. Dem steht auf der anderen Seite das Risiko gegenüber, daß sich dieser Aufwand nicht in einer entsprechend verbesserten Darstellung niederschlägt: der Mehrverbrauch an Rechenzeit kann zu Lasten des Reaktionsvermögens der Relevanzkugel gehen, unregelmäßige Datenobjekte sind insbesondere bei teilweiser Überlappung (ob nun rein optisch oder real) schwieriger voneinander zu trennen als solche von einheitlicher Form, und letztendlich besteht ein potentieller Konflikt zwischen den Ansprüchen des Faktors Form selbst – der eine gewisse minimale Größe und Detailstufe des abgebildeten Objekts voraussetzt, um seine Aussagekraft wirklich zu entfalten – und denen einer übersichtlichen Darstellung, die die Datenobjekte klein halten will, damit auch größere Mengen von ihnen in einem Browserfenster von im Allgemeinen begrenzter Größe noch so gut wie möglich optisch aufgelöst werden können.

Ein möglicher Kompromiß könnte dahin gehen, auf vollmodellierte flexible Datenobjekte zwar zu verzichten, aber die existierenden Objekte dennoch komplexer zu gestalten als bisher. Beispielsweise ließen sich die durch die angreifenden Kräfte hervorgerufenen Deformationen eventuell durch eigene separate *Teile* des Objekts modellieren – der `Extrusion`-Knotenprototyp böte sich etwa zur Erstellung von expliziten 'Beulen' an –, die der Grundform einfach nur überlagert und geeignet ausgerichtet würden. Oder es könnten von einem zentralen eigentlichen Datenobjekt einzelne Strahlen ausgehen, die ähnlich wie die zuerst angesprochenen Verbindungslinien auf die einzelnen relevanten Referenzobjekte zeigen und möglicherweise durch ihre Länge und/oder Dicke die Stärke der jeweiligen Beziehung zumindest grob andeuten würden. Auch hier sollte der jeweilige Aufwand gegen den zu erzielenden Nutzen abgewogen werden: Diese Lösungen sind für nur einzelne oder wenige Datenobjekte dem einfachen Verbindungslinienmodell nicht wirklich überlegen, während ihre Anwendung auf *alle* dargestellten Datenobjekte potentiell ähnliche Probleme wie der oben diskutierte Ansatz nach sich ziehen kann, wenn auch bei verringertem reinen Rechenaufwand.

4.4.4. Positionsbestimmung

Nach der Entscheidung bezüglich der geeigneten Form der Datenobjekte stellt sich als nächstes die Frage nach deren Positionierung und der Handhabung eventueller Kollisionen. Der vom Relevanzkugelmodell verwendete Ansatz basiert prinzipiell auf der Addition gewichteter Ortsvektoren, nämlich derer der einzelnen für ein Datenobjekt relevanten Referenzobjekte relativ zum Zentrum der Kugel, gewichtet sowohl mit den einzelnen referenzobjekteigenen Gewichts- als auch den datenobjektspezifischen Relevanzfaktoren. Das erste Problem hierbei ist, eine geeignete Normierung zu finden, damit der resultierende Positionsvektor nicht länger wird als der Kugelradius selbst und das Datenobjekt auf diese Weise das Kugelinnere verläßt. Ein denkbarer Ansatz besteht darin, der Relevanzkugel insgesamt einen vom Betrachter veränderbaren Dichtewert zuzuordnen, der ebenfalls in die Rechnung eingeht; auf diese Weise würden die Datenobjekte zwar an sich nicht am Verlassen der Relevanzkugel gehindert, aber der Betrachter könnte die Darstellung je nach Bedarf zur Laufzeit anpassen. Hierbei besteht freilich immer die Möglichkeit, daß der Betrachter bei hinreichend weiter Verteilung der Datenobjekte ein außerhalb der Kugel positioniertes Datenobjekt übersieht, weil es etwa auch den Bereich des Browserfensters selbst verlassen hat.

Eine andere Möglichkeit wäre, überlange Positionsvektoren einfach im Rahmen der Berechnung direkt auf eine feste Maximallänge zurechtzustutzen. Diese Methode ist freilich etwas brachial und könnte im schlimmsten Fall dazu führen, daß sich mehr Datenobjekte direkt an der Kugelschale versammeln, als dort aufgrund ihrer Relevanzwerte tatsächlich hingehören.

Letztlich gibt es noch die Möglichkeit, dafür zu sorgen, daß die Positionsvektoren von sich aus nie zu lang werden. Dies läßt sich beispielsweise einrichten, indem die *für die Darstellung verwendeten* Relevanzfaktoren (nicht zwangsläufig die Originalwerte) selbst so normiert werden, daß die Summe über alle Relevanzwerte eines beliebigen Datenobjekts aus der aktuellen Menge nie größer als 1 wird, d.h., indem man einmal das Maximum über alle diese Einzelsummen bildet und dann jeden einzelnen Relevanzwert durch den so erhaltenen Wert teilt. Damit kann, da auch die einzelnen Gewichtswerte immer im Bereich von 0 bis 1 liegen und alle Ortsvektoren von gleicher Länge sind, auch die Summe der Beiträge der einzelnen Ortsvektoren zum Gesamtbetrag des Positionsvektors maximal so groß werden wie die Länge eines *einzigen* Ortsvektors – und das ist gerade der Kugelradius.

4.4.5. Kollisionen

Kollisionen zwischen Datenobjekten können prinzipiell drei Ursachen haben, wobei wir hier unter 'Kollision' sowohl vollständige Überlagerungen als auch teilweise Verschmelzungen mehrerer Objekte verstehen wollen; für den letzteren Fall gelten die folgenden Punkte unter Vorbehalt entsprechender Näherungen.

- Die Addition an sich verschiedener Teilvektoren resultiert für zwei oder mehr Datenobjekte in ein und demselben Positionsvektor. Dieser Fall ist relativ harmlos, da er durch Manipulation eines oder mehrerer beteiligter Referenzobjekte, die zu einer entsprechenden Änderung der besagten Teilvektoren führt, vom Betrachter selbst behoben werden kann.

- Zwei oder mehr Datenobjekte haben exakt dieselben Relevanzwerte; dies kann insbesondere dann geschehen, wenn die Bandbreite an möglichen Relevanzwerten von vornherein klein ist. In diesem Fall werden die Objekte in der Darstellung zwangsläufig immer an derselben Stelle positioniert werden. Am ehesten läßt sich dies (unter Beibehaltung des Anspruchs auf eine möglichst breite Anwendbarkeit des Relevanzkugelmodells – es ist nicht unbedingt Sache der Relevanzkugel selbst, diese Entscheidung zu treffen) noch abfangen,

indem die externe Anwendung, die über die Ausgangsdaten verfügt, diese auf solche Übereinstimmungen hin überprüft und gegebenenfalls einem Daten*objekt* in der Darstellung mehrere Daten*sätze* zuordnet.

- Zwei oder mehr Datenobjekte haben teils identische, teils verschiedene Relevanzwerte, von denen aber aktuell nur die identischen zur Darstellung herangezogen werden, weil die Referenzobjekte, auf die sich die restlichen beziehen, gerade ein Gewicht von Null aufweisen. Dies kann potentiell zu einem echten Problem werden: VRML selbst kann solche Kollisionen nicht erkennen, die externe Anwendung müßte entsprechend implementiert und eng an das Relevanzkugelwerkzeug gekoppelt werden, um diesen Fall durch entsprechende Anpassungen behandeln zu können, und der Betrachter *will* unter Umständen die Gewichte der für ihn gerade irrelevanten Referenzterme nicht erhöhen und die möglicherweise für ihn gerade aussagekräftige Darstellung durch die erneute Trennung der einzelnen Objekte verkomplizieren. Nichtsdestoweniger gibt es auch für diesen Fall zumindest den Ansatz zu einer Lösung, der den Vorteil hat, potentiell auch den obigen zweiten Fall dann mit zu erfassen, wenn die externe Anwendung *keine* entsprechenden Sicherheitsabfragen vornimmt: Man sehe neben den bisher angesprochenen Interaktionsmöglichkeiten für Datenobjekte eine Methode vor, mit der nach Auswahl *eines* Datenobjekts ein Bereich definiert werden kann, der auch alle Nachbarn dieses Objekts innerhalb eines gegebenen Radius erfaßt, so daß diese mitmarkiert werden. Ein solcher Bereich wird zwangsläufig auch alle Datenobjekte miterfassen, die sich an derselben Position befinden wie der 'Anker', und obendrein ermöglicht diese Methode ganz allgemein eine Auswahl mehrerer dicht beieinanderstehender Datenobjekte auf einmal, ohne jedes einzelne mit dem Mauszeiger erfassen und anklicken zu müssen.

Eine letzte Möglichkeit, Kollisionen von Datenobjekten in der Darstellung zumindest für den Betrachter sichtbar zu machen, wenn auch nicht unbedingt aufzulösen, soll hierbei nicht unerwähnt bleiben: Prinzipiell erlaubt VRML ohne weiteres auch die Darstellung *transparenter* Objekte. Das heißt, es ist im Prinzip möglich, die einzelnen Datenobjekte mehr oder weniger durchscheinend zu gestalten, so daß sich zwei oder mehr überlappende Objekte klar durch ihren resultierenden relativen *Mangel* an Transparenz im Bereich der Überlappung bemerkbar machen würden. Allerdings ist nie ganz auszuschließen, daß diese Methode plattformbedingt daran scheitert, daß der VRML-Standard von Browsern als Mindestanforderung nur

verlangt, den Unterschied zwischen voller Transparenz (effektiver Unsichtbarkeit) und völliger Undurchsichtigkeit zu implementieren.

4.4.6. Interaktionen

Es bleibt abschließend noch zu klären, welche Möglichkeiten der Interaktion zwischen Betrachter und Datenobjekt sinnvoll erscheinen. Die Möglichkeit zur Hervorhebung einzelner Objekte wird vom Modell gefordert; in Anbetracht der in Abschnitt 4.4.1. angestellten Überlegungen bezüglich des Faktors Form bietet sich tatsächlich am ehesten eine Markierung durch Änderung der Farbe oder Helligkeit des Objekts als Reaktion auf einfaches Anklicken mit dem Mauszeiger an. Zusätzlich kann daran gedacht werden, ein Datenobjekt bereits bei einfacher *Berührung* durch den Mauszeiger farblich zu kennzeichnen und eventuell Zusatzinformationen einzublenden, solange der Kontakt erhalten bleibt; dies beispielsweise, damit sich der Betrachter vor einer eventuellen Markierung noch einmal vergewissern kann, daß er auch das richtige Objekt aus der Menge ausgewählt hat. Andererseits sollten die dieserart zur Verfügung gestellten Informationen zum aktuellen Objekt auch nicht ausufern, um die Übersichtlichkeit zu wahren und das Modell unabhängig von den konkret zugrundeliegenden Ausgangsdaten zu halten. Eine ausführliche Anzeige beispielsweise eines Textdokuments oder einer Filmdatei auf Anfrage ist eher eine Aufgabe für die externe Anwendung; die Relevanzkugel dient immer noch primär dazu, interessante Objekte erst einmal zu *finden*, damit mit ihnen dann anderweitig weitergearbeitet werden kann. Letztlich sollte in diesem Zusammenhang auch die schon in Abschnitt 4.4.5. angesprochene Methode zur Simultanauswahl mehrerer Objekte nicht vergessen werden.

4.5. Die Relevanzkugel als Gesamtheit

Während die ersten beiden Komponenten sich hauptsächlich mit der eigentlichen Darstellung befassen, geht es hier primär um die Einkapselung der einzelnen Mechanismen der Relevanzkugel gegenüber der Außenwelt und die Schnittstelle zur Interaktion mit derselben. Der Betrachter selbst interagiert mit der Relevanzkugel als Einheit eigentlich nur, wenn er sie als Ganzes rotieren läßt, was sich entweder bei geeignet ausgestatteten VRML-Browsern über deren Sichtsteuerfunktionen oder aber vermittels einer mit einem entsprechenden Sensor eigens definierten Komponente der Kugel realisieren läßt. Da weitestmögliche Plattformunabhängigkeit zur Aufgabenstellung gehört, empfiehlt sich letzteres.

Eine externe Anwendung, die auf die Relevanzkugel als Werkzeug zugreifen will, soll mit dieser möglichst einfach als Einheit arbeiten können, ohne sich mit den genauen Eigenarten der Relevanz- und Datenobjektknoten oder unnötiger Arbeit wie etwa der exakten Festlegung von Signalrouten befassen zu müssen; dies verringert gleichzeitig das Fehlerrisiko. Daher empfiehlt es sich, die einzelnen Eigenschaften der Referenz- und Datenobjekte nach außen hin in Form von einzelnen Werten oder Wertelisten auszudrücken und diese dann – über entsprechende Skriptfunktionen – bei Bedarf in die entsprechenden Parameter für die einzelnen Teilknoten zu konvertieren, beziehungsweise umgekehrt bei internen Änderungen in die Gegenrichtung übersetzte Ausgabeereignisse zu generieren. Ein- und Ausgänge sollten sich nur da auf Knoten beziehen, wo sinnvollerweise Standardknotentypen zur Anwendung kommen, beispielsweise für der Festlegung von optischen Eigenschaften durch `Material`-Knoten. Soweit die Anzahl der Objekte eines Typs nicht von vornherein bekannt und für die gesamte Lebensdauer einer Szene konstant ist, ist es außerdem sinnvoll, daß die entsprechenden Knoten erst zur Laufzeit vom Relevanzkugelskript generiert werden. Dies gilt insbesondere für die Referenzobjekte – je nach genauer Implementierung können die *Daten*-objekte wahlweise durch eine Vielzahl von Knoten oder einen einzigen mit komplexer Geometrie dargestellt werden, so daß im letzteren Fall diese Notwendigkeit entfällt.

Kapitel 5 : Implementierung

5.1. Überblick

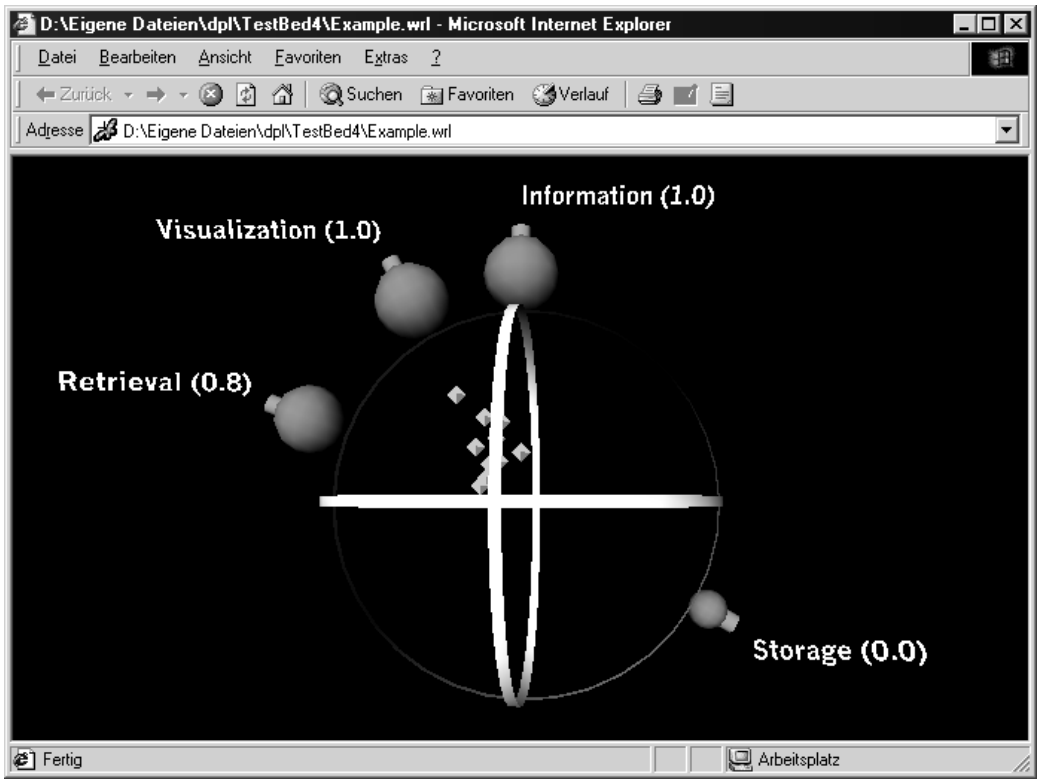

Abb. 17: Die VRML-Relevanzkugel

Die hier vorgestellte Implementierung definiert drei neue VRML-Knotenprototypen, die jeweils das einzelne Referenzobjekt, die Gesamtheit aller aktuellen Datenobjekte, und die Relevanzkugel als Ganzes modellieren; dazu kommt ein einfacher Hilfsprototyp zur leichteren internen Realisierung der Datenobjektlabels. Die Kenndaten der einzelnen Objekte – Namen, Gewicht, und Position (angegeben in Form einer Drehung um das Relevanzkugelzentrum aus einer vorgegebenen Ausgangsposition heraus) für Referenzobjekte, Namen, Relevanzen, und gegebenenfalls Markierungsstatus für Datenobjekte – werden dem Relevanzkugelknoten in Form von Wertelisten übergeben, wobei Werte an der jeweils gleichen Position in einer Liste sich auf dasselbe Objekt beziehen. Ausgenommen hiervon ist lediglich die Relevanzliste, da VRML keinen Datentyp für Listen *von* Listen vorsieht; statt dessen ergibt sich diese durch sukzessives Aneinanderhängen der Relevanzlisten der einzelnen Datenobjekte. Die Relevanzkugel kann sowohl selbständig unter Verwendung der ihr mitgegebenen Startwerte sofort nach Laden der sie enthaltenen Szene

aktiv werden als auch in inaktivem Zustand auf Eingaben von außen warten und erst auf Aufforderung starten. Einige der in Kapitel 4 beschriebenen Ideen haben Eingang in die Implementierung gefunden, so daß sie gegenüber dem Grundkonzept und den bekannten Vorgängerversionen durchaus gewisse funktionale Erweiterungen erfahren hat.

5.2. Das VRML-Gleichzeitigkeitsproblem

Der obige Begriff bezeichnet eine Art von Komplikation, die bei der Implementierung der hier vorgestellten Relevanzkugelversion gleich mehrfach auftrat und sich augenscheinlich direkt aus der VRML-Spezifikation ergibt (siehe [VRML 97], speziell Abschnitte 4.10, *Event Processing*, 4.12, *Scripting*, und B.5.3, *Sending eventOuts or eventIns*), und zwar wie folgt:

1.) Eine Eigenschaft des VRML-Zeitmodells ist, daß von einem Knoten ausgehende Ereignisse, die als Reaktion auf ein an ihm eingehendes Ereignis erfolgen, den Zeitstempel dieses ursprünglichen Ereignisses erben. Das kann im Extremfall zu einer Kette oder Kaskade von aufeinander folgenden Ereignissen führen, die die VRML-Spezifikation alle als gleichzeitig auffaßt (auch wenn sie in der Praxis nacheinander abgearbeitet werden).

2.) Wird während einer Ausführung eines Java-Knotenskripts mehrfach versucht, ein und demselben Signalausgang einen Wert zuzuweisen, so führt nur der erste solche Versuch tatsächlich zu einem ausgehenden Ereignis; alle weiteren werden kurzerhand ignoriert.

3.) Da nun Routen in VRML-Szenen prinzipiell zwischen beliebigen Signalein- und -ausgängen des gleichen Typs gezogen werden können, kann es geschehen, daß ein Knoten A *mehrere* Ereignisse mit identischem Zeitstempel erhält – beispielsweise von mehreren anderen Knoten, die alle dasselbe Ausgangsereignis empfangen und reagiert haben –, die alle eine Reaktion erfordern, die ihrerseits einen bestimmten Ausgang von A anspricht, um weitere Ereignisse zu generieren. Da aber alle diese Ereignisse als gleichzeitig interpretiert werden, greift die unter 2.) beschriebene Regel und nur die allererste Reaktion führt tatsächlich zu einem ausgehenden Ereignis, während alle anderen einschließlich eines eventuellen Endergebnisses, das erst durch Auswertung *aller* eingehenden Ereignisse ermittelt werden kann, praktisch verschluckt werden.

Nach der VRML-Spezifikation müßte sich das Problem durch geeignetes Multithreading im Prinzip beseitigen lassen – jedoch ist nicht klar, ob tatsächlich jeder VRML-Browser diese Unterscheidung korrekt handhabt. Statt dessen verwendet die hier vorgestellte Implementierung auf eher konventionelle Weise interne Zähler und gelegentliche Synchronisationssignale, um sicherzustellen, daß ein Signalausgang erst dann angesprochen wird, wenn sein endgültiger Wert tatsächlich festliegt. Diese Version funktioniert unabhängig von der Multithreading-Fähigkeit des darstellenden Browsers auf jeden Fall, macht allerdings auch die Ansteuerung der Relevanzkugel von außen etwas umständlicher, indem sie vom Anwender ein zusätzliches Bestätigungssignal für das Ende der Eingabe verlangt und erwartet, daß nach Eingang dieses Signals keine weiteren Eingaben und/oder Anfragen erfolgen, bis die Kugel selbst durch ein Gegensignal anzeigt, daß alle internen Aktualisierungsvorgänge abgeschlossen sind.

5.3. Der Referenzobjekt-Knoten

Im Rahmen dieser Implementierung setzt sich der Referenzobjekt-Knotenprototyp im wesentlichen aus folgenden Komponenten zusammen:

- Den Geometrie- und Materialknoten, die Form und Farbgebung seiner einzelnen Teile – Basiskugel, zylinderförmiger Bedienungsknopf, und eine Zeichenkette als Label – festlegen.

- Einer Reihe von Transformknoten zur Positionierung und Skalierung sowohl des Referenzobjekts als Ganzem als auch der Komponenten relativ zueinander. Diese Knoten sorgen beispielsweise dafür, daß das Referenzobjekt glatt auf der Relevanzkugelschale (modulo eventuellem Zusatzabstand) aufsitzt, ohne auch bei Größenänderungen ins Kugelinnere einzudringen oder unbeabsichtigt abzuheben, und daß der Bedienungsknopf sich nicht aus dem Kugelteil löst oder in diesen einsinkt. In diese Gruppe gehört im weiteren Sinne auch ein `Billboard`-Knoten, der dafür sorgt, daß das Referenzobjektlabel unabhängig von Position und Ausrichtung des Objekts oder der Relevanzkugel immer im rechten Winkel zur Blickrichtung des Betrachters angezeigt wird.

- Sensorknoten, die auf Mausereignisse über der Objektgeometrie reagieren und sie, gegebenenfalls schon in geeigneter Form in Bewegungen übersetzt, an die anderen Knoten weiterleiten.

- Letztlich dem Skriptknoten, der neben der Initialisierung des Objekts und interner Verwaltungsarbeit auch für die Ausgabe sämtlicher für die Arbeit des Referenzobjektknotens als Teil der Relevanzkugel nötigen Ereignisse verantwortlich zeichnet. Insbesondere übernimmt das Skript die Aufgabe, aus den aktuellen Rotations- und Translationswerten des Referenzobjekts die Projektion von dessen Position auf die Relevanzkugelschale vorzunehmen und das Ergebnis, welches dann zur Positionsbestimmung der Datenobjekte herangezogen wird, zur einfacheren Weiterverwertung sofort in kartesische Koordinaten umgerechnet auszugeben.

Von der Gestaltung her orientiert sich diese Art von Referenzobjekt stark am historischen Vorbild, bietet allerdings als standardmäßig eingeschaltete zusätzliche Option eine explizite numerische Gewichtsanzeige als Bestandteil des Labels, die in Klammern hinter dem eigentlichen Bezeichner eingeblendet wird.

Da in dieser Implementierung die Referenzobjekte jeweils erst zur Laufzeit vom Relevanzkugelskript aus den vorgegebenen Werten und Eigenschaften generiert werden, ist der Knotenprototyp möglichst einfach gehalten. Er nimmt, mit Ausnahme von Materialknoten zur Festlegung seiner optischen Eigenschaften unmittelbar *nach* seiner Erzeugung, keinerlei Ereignisse von außen an, sondern produziert ausschließlich eigene Ausgaben als Reaktion auf Interaktionen des Betrachters mit dem Objekt. Selbst diese bestehen in erster Linie aus der Ausgabe seiner einmalig vergebenen internen Nummer – die sich direkt aus der Position der dieses Objekt definierenden Werte in den einzelnen Ausgangslisten ergibt und die Zuordnung eines eintreffenden Ereignisses zu einem bestimmten Referenzobjekt überhaupt erst ermöglicht – als Signal, daß sich an diesem Objekt überhaupt irgend etwas getan hat; weitergehende Informationen können und müssen die Empfängerknoten selbständig von seinen Signalausgängen abfragen. Dies mag umständlich klingen, spart aber in der Praxis Signalrouten und damit Verwaltungsaufwand für den VRML-Browser.

5.4. Der Datenobjekt-Knoten

Bereits früh in der Entwicklung hat sich der naive Ansatz, einen Knotenprototyp für das *einzelne* Datenobjekt zu entwerfen, diesen mit der nötigen Funktionalität zu versehen, und dann für die konkrete Relevanzkugeldarstellung einfach so viele von diesen Knoten zu verwenden wie nötig, als praxisfern herausgestellt, da er mit

zunehmender Anzahl der Datenobjekte schnell zu einer nicht mehr tolerierbaren Verlangsamung der Darstellung führt. Die Ursache dafür ist darin zu suchen, daß eine solche Implementierung – neben dem reinen Verwaltungs- und Renderingaufwand für eine Vielzahl von Einzelknoten – natürlich auch eine Vielzahl von Signalrouten innerhalb des Szenengraphen erfordert, damit sichergestellt ist, daß jedes Datenobjekt auch tatsächlich über Ereignisse an jedem potentiell relevanten Referenzobjekt informiert wird. Das führt dazu, daß bei größeren Datenobjektmengen eine Vielzahl von Kopien ein und desselben Signals weitergeleitet und jeweils individuell von den Skripten der einzelnen Empfängerknoten ausgewertet werden müssen; dadurch entsteht aber auf handelsüblichen Rechnersystemen in Ermangelung von entsprechend vielen parallel arbeitenden Prozessoreinheiten ein Flaschenhals, da die Kopien des Originalereignisses de fakto eine nach der anderen weitergereicht und die sie auswertenden Skripte ebenso sequentiell aufgerufen und ausgeführt werden müssen. Hinzu kommt, daß natürlich auch die Skripte ihrerseits Ereignisse generieren, um die Darstellung ihrer Objekte anzupassen, und diese Ereignisse auf dem Weg zu ihren Zielknoten denselben Flaschenhals passieren.

Aus diesem Grund verwendet die tatsächliche Implementierung einen einzigen Datenobjektknoten, der fester Bestandteil der Relevanzkugel ist und praktisch alle Aufgaben handhabt, die im Zusammenhang mit den einzelnen Datenobjekten anfallen könnten. Die Form der Datenobjekte selbst bleibt – der einfacheren Modellierbarkeit und der mit geringerem Rechenaufwand verbundenen besseren Reaktionsfähigkeit der Darstellung gerade bei großen Objektmengen wegen – einheitlich und fest, wenn auch die quaderförmigen Objekte der Vorgängerversionen durch Oktaeder ersetzt wurden, die bei gleichen Dimensionen ein um zwei Drittel verringertes Volumen aufweisen und so eine bessere optische Trennung bei hoher Packungsdichte erlauben. Die Hervorhebung aktuell markierter und/oder berührter Datenobjekte erfolgt primär farblich, bei entsprechend gesetzten Optionen zusätzlich durch das Einblenden der mit den Objekten assoziierten Labels in der jeweils gleichen Farbe.

Um die Zusammenhänge zwischen Referenz- und Datenobjekten besser zu verdeutlichen, verwendet die implementierte Version zwei der in Abschnitt 4.4.3. angesprochenen Methoden, die beide in erster Linie vom Datenobjektknoten behandelt werden. Erstens wird, sobald der Betrachter ein Referenzobjekt mit der Maus erfaßt, ein Signal ausgelöst, welches das Datenobjektskript anweist, sämtliche Datenobjekte, für die dieses Referenzobjekt relevant ist, farblich hervorzuheben;

dabei wird auf die bereits definierten Farben für als berührt zu kennzeichnende Objekte zurückgegriffen, auch wenn dieser Fall in jeder anderen Hinsicht nicht als normale Berührung behandelt wird und beispielsweise nicht zur Einblendung der Datenobjektlabels führt. Zweitens werden, wenn der Betrachter ein *Daten*objekt berührt, für die Dauer dieser Berührung entsprechend eingefärbte Linien eingeblendet, die das Objekt mit den relevanten Referenzobjekten verbinden. (Prinzipiell könnte diese zweite Methode auch beim Erfassen eines Referenzobjekts zur Anwendung kommen; allerdings enthält die als Normalfall angenommene Relevanzkugel wesentlich mehr Daten- als Referenzobjekte, so daß die resultierende Anzahl von Verbindungslinien potentiell mehr verdecken als illustrieren würde.)

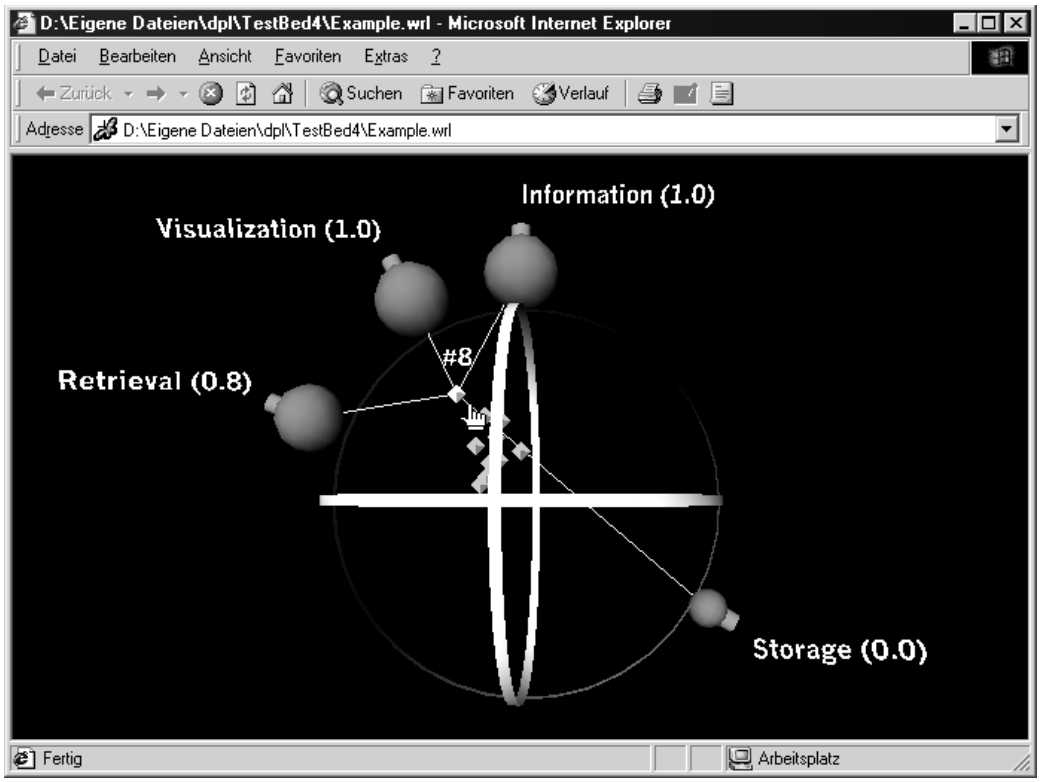

Abb. 18: Verbindungslinien zum berührten Datenobjekt

Wesentliche Bestandteile des Datenobjektknotens sind:

- *Ein* Geometrieknoten, der alle eigentlichen Datenobjekte modelliert. Dabei handelt es sich um einen VRML-Standardknoten vom Typ IndexedFaceSet, der dazu dient, komplexe Formen aus einzelnen Polygonen zusammenzusetzen und auch gleich die Möglichkeit bietet, die einzelnen Flächen individuell einzufärben – was sich natürlich zur Markierung einzelner Objekte nutzen läßt.

- Zwei an diesen Geometrieknoten gekoppelte Sensorknoten zum Abfangen von Mausereignissen. Von diesen dient einer primär zum Erkennen von Kontakten des Mauszeigers mit der dargestellten Geometrie, die das Datenobjektskript dann als Berührung am der Kontaktstelle am nächsten positionierten Objekt interpretiert. Der andere erkennt sowohl Mausklicks über der Geometrie als auch Drag-Ereignisse, und arbeitet mit dem Selektorgeometrie- und dem Skriptknoten zusammen, um die Auswahl und Markierung von Objekten zu ermöglichen.

- Ein Gruppenknoten, der je nach Bedarf die darzustellenden Datenobjektlabels aufnimmt, wobei diese zur leichteren Handhabung über einen einfachen Hilfsprototypen definiert sind. Jedes einzelne Label ist also für sich ein Knoten eben dieses Hilfstyps, wird aber ausschließlich vom Datenobjektknotenskript definiert, positioniert, und gegebenenfalls wieder gelöscht; die einzige Eigenfunktionalität, die sie aufweisen, besteht wie bei den Labels der Referenzobjekte in einem internen Billboard-Knoten zur Ausrichtung des Schriftzugs hin zum Betrachter.

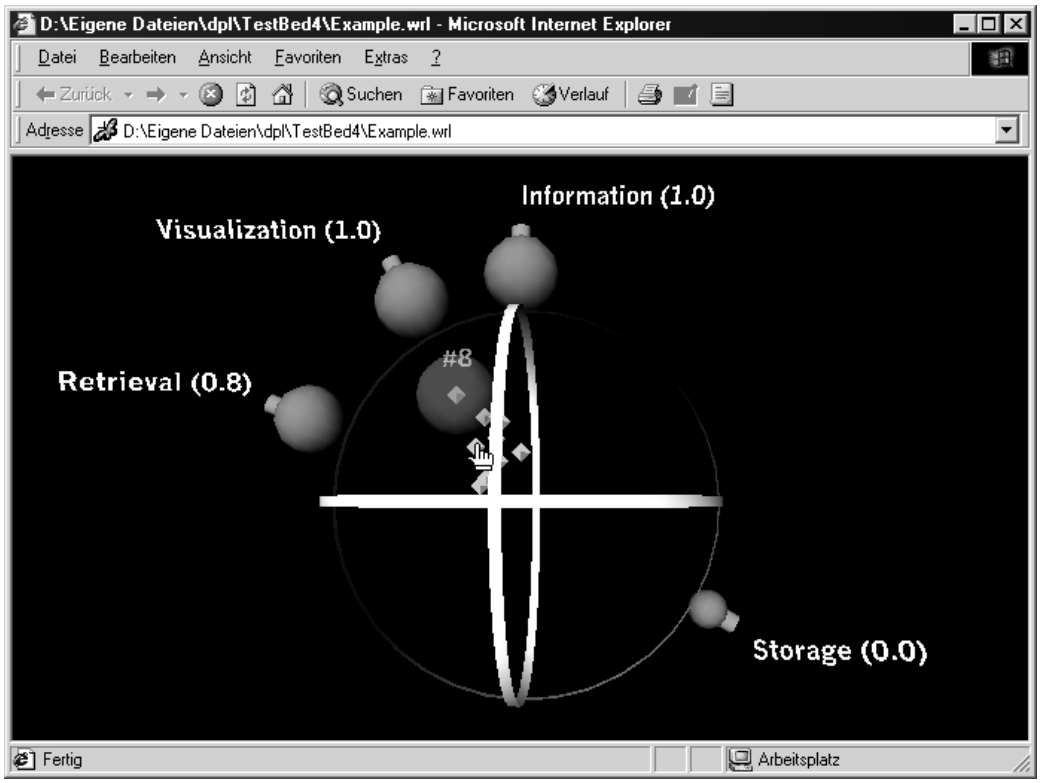

Abb. 19: Der Mehrfachauswahl-Selektor

- Ein Geometrieknoten vom Standardtyp IndexedLineSet, der – so diese Option aktiviert ist – bei Berührung eines Datenobjekts mit dem Mauszeiger die Verbindungslinien zwischen diesem und den relevanten Referenzobjekten modelliert.

- Der Selektor zur potentiellen Auswahl mehrerer Datenobjekte in der Nähe eines als Ankerpunkt zuerst durch Mausklick ausgewählten (siehe Abb. 19). Hierbei handelt es sich einfach um eine vom Knotenskript frei positionier- und skalierbare transparente Kugel, deren Größe je nach Bewegung des Mauszeigers bei gedrückt gehaltener Maustaste variiert, bis durch Wiederloslassen der Taste alle gerade ganz oder teilweise von ihr erfaßten Datenobjekte aus- beziehungsweise abgewählt werden, je nachdem, was für das zuerst erfaßte Objekt galt.

- Schließlich der (einzige) Skriptknoten selbst. Dieser fällt in Anbetracht aller Aufgaben, die er handhaben muß, zwangsläufig komplex aus, hilft aber gerade dadurch, das zu Beginn angesprochene Flaschenhalsproblem zu vermeiden, indem zu jeder Zeit nur relativ wenige Ereignisse zwischen einzelnen Knoten in der Szene ausgetauscht werden müssen.

Aufgrund der unterschiedlichen Aufgaben, die der Datenobjektknoten praktisch an zentraler Stelle der Relevanzkugel übernimmt, gestaltet sich auch seine Schnittstelle zum Rest der Szene – und indirekt zur Außenwelt – vergleichsweise umfangreich. Für die reine Funktionalität genügen im Prinzip Felder zur Aufnahme der Relevanz- und Identifizierungsdaten sowie Möglichkeiten zum Empfang von Signalen der Referenzobjekte sowie zur Ausgabe von Ereignissen, wenn Datenobjekte berührt oder markiert werden; dazu kommen jedoch, da die Relevanzkugel ja möglichst allgemein verwendbar bleiben soll, auch noch Felder zur Definition beispielsweise der Größe und Beschaffenheit der einzelnen Elemente der Darstellung sowie Signalein- und -ausgänge, die den nötigen Zugriff auf eben diese Felder bieten. Gerade bei der Anpassung der Darstellung, etwa um neue eingehende Daten zu visualisieren, kann sich hier auch potentiell das Gleichzeitigkeitsproblem am ehesten bemerkbar machen, da in dieser Situation durchaus einmal mehrere verschiedene potentiell simultan eintreffende Ereignisse Auswirkungen etwa auf die Positionierung der Datenobjekte haben können. Soweit es sich um rein relevanzkugelinterne Ereignisse wie etwa eine Serie eintreffender Signale von einer gerade neu erstellten Menge von Referenzobjekten handelt, läßt sich dieser Fall über

interne Zähler handhaben. Bei Informationen von außerhalb dagegen kann das Datenobjektskript grundsätzlich nicht wissen, nach wie vielen vom VRML-Browser potentiell als gleichzeitig interpretierten Ereignissen die Eingabe als abgeschlossen betrachtet werden kann; daher benötigt der Knoten außerdem einen Eingang für ein Update-Signal, welches ihm genau dies mitteilt, und ebenso einen Ausgang für das entsprechende Bestätigungssignal, da unabhängig von der Betrachtungsweise des Browsers im Rahmen einer eventuell umfangreicheren Updateaktion immer etwas Zeit verstreichen wird.

5.5. Der Relevanzkugel-Knoten

Dieser Knoten dient primär dazu, die einzelnen Objektknoten gegenüber der Außenwelt zu einer Einheit zusammenzufassen und eventuell anfallende interne Verwaltungsarbeit zu handhaben, die von den bisherigen Knotentypen nicht erfaßt wird. Dazu gehört primär die Verwaltung der Referenzobjektknoten: ihre Erzeugung, ihre Einbindung in das relevanzkugelinterne Signalnetzwerk, das Weitergeben von potentiell interessanten Referenzobjektereignissen nach außen, und schließlich auch bei Bedarf deren saubere Entsorgung.

Der Knotenprototyp setzt sich im wesentlichen aus folgenden Komponenten zusammen:

- Einem fest eingebundenen Datenobjektknoten, der seine Arbeit weitgehend selbsttätig verrichtet. Tatsächlich sind viele Felder sowie Ein- und Ausgänge des Relevanzkugelknotens nichts anderes als Durchgänge an ihre Entsprechungen am Datenobjektknoten – eine an ersterem neu eingetroffene Relevanzliste beispielsweise wird direkt an letzteren weitergereicht, ohne das Relevanzkugelskript im geringsten zu belasten, und ebenso geht ein Signal, daß ein neues Datenobjekt markiert worden ist, unmittelbar nach außen.

- Einem einfachen Gruppenknoten, der die zur Laufzeit erzeugten Referenzobjektknoten als Kinder aufnimmt.

- Knoten zur Darstellung der verschränkten Ringelemente, die die Relevanzkugelschale symbolisieren. Eine solide, wenn auch transparente, Kugelform verbietet sich von selbst, da sie unter VRML die Interaktion des Betrachters mit allen hinter ihr bzw. in ihrem Inneren befindlichen Objekten unterbinden würde; andererseits hinterläßt eine rein implizite Andeutung der Kugelform allein auf

der Basis der Anordnung und Ausrichtung der Referenzobjekte leicht ein etwas merkwürdiges Gefühl beim Betrachter. Im Fall dieser Implementierung haben die Ringelemente jedoch daneben noch einen praktischen Nutzen: mit einem Sensor versehen und an einen die gesamte Kugeldarstellung umfassenden `Transform`-Knoten gekoppelt, funktionieren sie als Bedienungselement, an dem der Betrachter die Relevanzkugel als Ganzes erfassen und mit dessen Hilfe er sie drehen kann.

- Letztlich dem unvermeidlichen Skriptknoten, der primär die oben angeführte Verwaltungsarbeit in Bezug auf die Referenzobjekte verrichtet und von diesen eintreffende Signale bei Bedarf nach außen hin weiterleitet.

Das *Interface* des Relevanzkugelknotens besteht in der Hauptsache aus Feldern, die die Eigenschaften der Kugel und aller einzelnen Objekte beschreiben, sowie aus den dazugehörigen Signalein- und -ausgängen zum Austausch von Informationen mit der Außenwelt. Dazu kommen noch eine Handvoll von Ein- und Ausgängen zur besonderen Verwendung, etwa zur Entgegennahme und Bestätigung von Update-Signalen und zur Weiterleitung von Informationen über vom Betrachter ausgelöste kugelinterne Ereignisse.

5.6. Initialisierung, Aktualisierung und Signalfluß

Zu Anfang ist die geplante Relevanzkugel immer unsichtbar. Zwar sind unter Umständen aus der ursprünglichen Quelldatei oder aufgrund der prototypeigenen Voreinstellungen bereits Werte zu ihrer Definition vorgegeben, aber es existieren noch keine Referenzobjektknoten, das Datenobjektskript hat noch keine darzustellenden Flächen definiert, und selbst die Ringelemente haben noch ihren Ursprungsradius von gerade Null, sind also ebenfalls nicht zu sehen.

All dies ändert sich mit dem Eintreffen des ersten Update-Signals. Je nach *auto-Start*-Einstellung der Relevanzkugel löst sie dieses entweder selbst aus oder wartet darauf, daß sie eines von einer geeigneten externen Quelle erhält; in letzterem Fall hat diese Quelle die Möglichkeit, eigene Werte in die entsprechenden Eingänge der Kugel zu schreiben, bevor die erste Darstellung auf dem Bildschirm erfolgt. Ist das Signal erst einmal eingegangen, so beginnt ein umfangreicher Prozeß, in dessen Rahmen zunächst einmal sämtliche bisher zwischengespeicherte Werte tatsächlich übernommen werden. Das Relevanzkugelskript konstruiert aus den zur Beschreibung der Referenzobjekte herangezogenen Werten und Wertelisten einen String in

VRML-Syntax, der genau diese Objekte definiert – an diesem Punkt werden diesen auch ihre individuellen Nummern zugeordnet –, und übersetzt diesen vermittels der in [VRML 97], Punkt 4.12.10.9 und Abschnitte B.6.5 bzw. B.9.2.1 beschriebenen *createVrmlFromString()*-Methode in ein Feld neu generierter Referenzobjektknoten. Anschließend etabliert es die nötigen Routen zwischen deren Signalaus- und seinen eigenen Eingängen und reicht sie dann als Kinder an den kugelinternen Gruppenknoten weiter, wobei eine Kopie des Feldes zu Referenzzwecken an den Datenobjektknoten geht, der seine eigenen Signalrouten entsprechend einrichtet; daneben sendet es ein zusätzliches Warnsignal an den Datenobjektknoten, das diesen auffordert, seinen internen Zähler zurückzusetzen, weil das Eintreffen einer Reihe von gleichzeitigen Referenzobjektmeldungen bevorsteht[1]. Nun beginnen die Referenzobjektknoten mit ihrer jeweiligen eigenen Initialisierung, an deren Abschluß das Senden ihrer ersten Ereignissignale steht. Diese Signale gehen über die gezogenen Routen primär an den Datenobjektknoten, der daraufhin die Positionen und Gewichte der sich meldenden Referenzobjektknoten aus deren *position_changed*- und *weight_changed*-Ausgängen liest und die nötigen Anpassungen an den Positionen seiner inzwischen definierten Datenobjekte vornimmt. Dabei zählt er mit, wie viele Meldungen er bereits erhalten hat, um die tatsächliche *Darstellung* auch wirklich erst anzupassen, wenn alle Referenzobjektknoten fertig initialisiert sind, und damit das Gleichzeitigkeitsproblem zu umgehen. Ist dies geschehen, so gibt der Datenobjektknoten seinerseits eine Bestätigung an den Relevanzkugelknoten weiter, damit dieser den Abschluß der Updateoperation nach außen signalisieren kann.

Zur eigentlichen Laufzeit der Szene, während der lediglich der Betrachter mit der Darstellung interagiert, gestalten sich die Abläufe wesentlich einfacher. Das Gleichzeitigkeitsproblem fällt nicht ins Gewicht, da der Betrachter immer nur auf eine Komponente auf einmal einwirken kann. Im wesentlichen werden Ereignisse von den einzelnen Sensoren der Kugel als Reaktion auf den Kontakt des Mauszeigers mit den entsprechenden Bestandteilen der Darstellung, teilweise auch erst nach Betätigung der Maustaste oder Ziehen bei gedrückter Taste, generiert und in den sie empfangenden Skripten ausgewertet, worauf diese wiederum entsprechende eigene Reaktionen in Gang setzen; die auf diese Weise entstehenden Ereigniskas-

[1] Im Prinzip ist dieses Warnsignal ein Relikt aus einer früheren Version, die nicht automatisch bei jedem Update an den Referenzobjekten auch neue *Knoten* erzeugte, und heute theoretisch überflüssig; es hat sich jedoch nicht als schädlich erwiesen und wurde daher belassen.

kaden sind eher kurz. Beispielsweise geschieht bei Berührung eines Datenobjekts mit dem Mauszeiger normalerweise, d.h., wenn alle Optionen tatsächlich verfügbar und eingeschaltet sind, folgendes:

- Der an den Datenobjekt-Geometrieknoten gekoppelte TouchSensor-Knoten registriet den Kontakt und den aktuellen Kontakt*punkt*.

- Das Datenobjektskript vergleicht die Koordinaten mit allen aktuellen Datenobjektpositionen und identifiziert so das tatsächlich berührte Objekt, nämlich das am nächsten befindliche.

- Soweit dieses Datenobjekt vorher noch nicht berührt wurde und entsprechende Farben in *dataColor* definiert sind, werden seine Seitenflächen neu eingefärbt; es sind normalerweise für diesen Fall zwei Farben definiert, wobei sich die Entscheidung, welche verwendet wird, danach richtet, ob das Datenobjekt bereits markiert ist oder nicht.

- Gleichzeitig wird der dem Datenobjekt zugeordnete Labelknoten in die Darstellung eingeblendet, wobei er die aktuelle Farbe des Datenobjekts selbst übernimmt. (Die Position des Labels wird ständig zusammen mit der des Datenobjekts an sich aktualisiert, aber der Knoten selbst ist nicht ständig Bestandteil des Szenengraphen.)

- Ebenso werden Verbindungslinien in derselben Farbe zwischen der Datenobjektposition und den physischen Mittelpunkten der Kugelteile der für das Datenobjekt relevanten Referenzobjekte – nicht deren zur Datenobjektpositionsbestimmung herangezogenen 'virtuellen' Positionen auf der Relevanzkugelschale! – definiert und eingeblendet.

- Letzlich generiert das Datenobjektskript noch ein Signal, welches die Nummer des zuletzt berührten Datenobjekts enthält, und merkt sich diese, um beim Eintreffen weiterer Berührungsereignisse entscheiden zu können, wann tatsächlich ein neues Datenobjekt kontaktiert worden ist. Dieses Signal hat relevanzkugelintern keine Bedeutung, kann aber bei Bedarf von externen Knoten oder Anwendungen empfangen und ausgewertet werden.

Andere Ereignisse werden im Prinzip ähnlich gehandhabt. Bei Manipulation eines Referenzobjekts durch den Betrachter beispielsweise wird dessen Position bzw.

Gewicht und Darstellung angepaßt, gleichzeitig gehen Signale heraus, die Datenobjekt- und Relevanzkugelskript über die Änderung informieren, so daß diese entsprechend reagieren können – das Datenobjektskript durch Anpassung der Datenobjektpositionen, das Relevanzkugelskript durch die Generierung neuer potentiell nach außen gehender Ereignisse, die die aktuellen Positions- und Gewichtswerte für alle Referenzobjekte enthalten.

Bezüglich solcher Interaktionen mit der Außenwelt existiert bei dieser Implementierung eine eindeutige Richtungsabhängigkeit. Signale aus dem Relevanzkugelinneren nach außen erfahren keine Verzögerung; es kann höchstens einmal geschehen, daß während eines gerade ablaufenden Update-Prozesses die Signalausgänge des Relevanzkugelknotens noch nicht die aktuellen Werte erhalten haben, die sie bis zum Abschluß zugewiesen bekommen werden. Andererseits gestaltet sich die Ansteuerung der Relevanzkugel durch von außen an sie herangetragene Ereignisse weniger dynamisch; da die Kugel wegen des Gleichzeitigkeitsproblems gar nicht erst versucht, solche Ereignisse sofort bei deren Eintreffen abzuarbeiten, andererseits aber auch von sich aus nicht wissen kann, wann dies sicher möglich ist, muß jedes solche Ereignis bzw. Folge von Ereignissen mit einem Update-Signal abgeschlossen werden, welches dann seinerseits naturgemäß auch wieder einen vollständigen Updateprozeß auslöst.

Kapitel 6 : Bewertung und Ausblick

6.1. Prinzipielle Vor- und Nachteile des Relevanzkugelmodells

Aufgrund der historischen Herleitung der Relevanzkugel aus dem VIBE-Modell haben sich bisherige Betrachtungen der Vorteile des Modells im Wesentlichen auf einen direkten Vergleich der beiden beschränkt. Die beiden Hauptvorteile, die sich dabei ergeben, sind wie folgt:

- Durch den Übergang von zwei zu drei Darstellungsdimensionen wird Raum gewonnen, so daß zum einen mehr Objekte und Referenzpunkte Platz finden und zum anderen die Wahrscheinlichkeit, daß zwei Objekte ohne näheren Bezug zueinander nahe beieinander positioniert werden, von vornherein deutlich reduziert wird ([Hemmje 99], S.137ff).

- Durch die hinzugewonnenen Interaktionsmöglichkeiten lassen sich auch eventuell noch verbliebene Konflikte relativ leicht auflösen und die Darstellung gewinnt allgemein an Flexibilität.

Betrachtet man die Relevanzkugel in dem Kontext, in dem sie ursprünglich entworfen wurde – d.h., als Hilfsmittel zur Dokumentsuche, speziell zum schnellen Aussondern einer relativ kleinen Anzahl von für den Anwender wirklich interessanten Dokumenten aus der Gesamtmenge der Ergebnisse einer Suchanfrage –, so ist sie für den geübten Anwender bei geeigneter Implementierung tatsächlich recht einfach bedienbar. Die Suche nach relevanten Objekten reduziert sich im Prinzip darauf, die gerade interessanten Referenzobjekte hoch zu gewichten (im Vergleich zu den im Moment nicht interessierenden, deren Gewicht sich bis auf Null herunterregeln läßt) und nahe beieinander zu positionieren, so daß ihre kombinierte 'Anziehungskraft' gerade die gesuchten Datenobjekte am weitesten aus der Kugelmitte heraus in ihre Richtung zieht und damit eindeutig erkennbar macht. Für eine genauere Analyse der Zusammenhänge können bei Bedarf einzelne Datenobjekte markiert und dann deren Reaktionen bei weiterer Veränderung der Referenzobjektgewichte beobachtet werden. Soweit die Bedienbarkeit der Implementierung nicht zu umständlich ausfällt, lassen sich so relevante Objekte recht schnell und bequem aus einer größeren Menge herauspicken.

Jedoch hat auch die Relevanzkugeldarstellung ihren Anteil an potentiellen Problemen:

- In der Praxis relativiert sich der durch die Hinzunahme der dritten Dimension gewonnene Vorteil schnell wieder durch den technisch bedingten Mangel an *echten* 3D-Abbildungsmöglichkeiten. Standardmäßig wird eine dreidimensionale Szene doch wieder nur auf den zweidimensionalen Bildschirm projiziert und der Eindruck von Tiefe zusätzlich unter Ausnutzung des räumlichen Vorstellungsvermögens des Betrachters mit verschiedenen optischen Tricks vorgetäuscht; im Fall der Relevanzkugel, die je nach genauer Implementierung für diese Täuschung unter Umständen nur relativ wenige brauchbare Anhaltspunkte bietet (da sie konzeptionell eher im leeren Raum schwebt und keine gestalterischen Elemente brauchen kann, die den Blick oder auch nur Zugriff auf die einzelnen Objekte verstellen), kann dies für den menschlichen Betrachter gegebenenfalls eine zusätzliche bewußte Anstrengung und damit eher eine Belastung als eine echte Hilfe bedeuten.

- Die Darstellung ist zunächst einmal nicht wirklich intuitiv – weder orientiert sie sich an vertrauten Vorbildern, noch gibt es eine unmittelbar offensichtliche Verbindung zwischen den einzelnen dargestellten Objekten. Daraus folgt, daß ein Betrachter, der dem Modell zum ersten Mal begegnet, auf jeden Fall Zeit benötigt, um eventuell mitgelieferte Erklärungen aufzunehmen, sich durch Ausprobieren mit den Interaktionsmöglichkeiten vertraut zu machen, und das Erkennen von Zusammenhängen (also in erster Linie das Arrangieren von Referenzobjekten so, daß die subjektiv interessanten Datenobjekte hervorstechen) zu üben.

- Die je nach genauer Implementierung mehr oder weniger stark betonte Kugelform der Darstellung zieht den Blick des Betrachters zunächst einmal unwillkürlich in Richtung der Kugelmitte; andererseits liegen Datenobjekte mit hoher Relevanz mit erhöhter Wahrscheinlichkeit näher an der Kugel*schale*, wo sich auch die Referenzobjekte als primäre Bedienungselemente befinden. Mit anderen Worten, die Konstruktion neigt dazu, die Aufmerksamkeit des Betrachters zumindest auf den ersten Blick von den ihn interessierenden Details *abzulenken*, statt sie hervorzuheben. Im Prinzip läßt sich dieses Problem zumindest teilweise korrigieren, wenn man von vornherein bewußt möglichst zusammenhängende Gruppen von nahe beieinander stehenden Referenzobjekten bildet und auf eine gleichmäßige Verteilung über die Kugelschale verzichtet; auf diese Weise erhöht sich die Wahrscheinlichkeit, von Anfang an aussagekräftige Cluster von Datenobjekten zu erhalten, die dann wieder die Aufmerksamkeit

des Betrachters auf sich (und damit vom Kugelschwerpunkt weg) ziehen können.

- Zwangsläufig kann das Modell schlecht zwischen mehreren Datenobjekten unterscheiden, die aus welchem Grund auch immer an derselben Stelle positioniert werden. Abschnitt 4.4. stellt verschiedene Ansätze vor, die sich mit diesem Problem befassen; eine endgültige Lösung würde jedoch eine sichere Methode voraussetzen, derartige Kollisionen als solche zu erkennen und zu behandeln, sie aber auch gleichzeitig von Fällen zu unterscheiden, in denen zwei nahe beieinander positionierte Datenobjekte aufgrund ihrer Größe zwar teilweise miteinander verschmelzen, aber bei geeigneter Wahl des Berührungspunktes immer noch separat angesprochen werden können.

6.2. Bewertung der vorliegenden Implementierung

Im Rahmen der Implementierung wurde zu Testzwecken ein Java-Applet erstellt, welches eine Textdokument-Stichwortsuche nach Schlüsselbegriffen erlaubt, eine einfache Relevanzbewertung je nach genauer Fundstelle vornimmt, und mit einer auf der gleichen HTML-Seite eingebetteten, anfangs leeren Relevanzkugel kommuniziert, um die Suchergebnisse zu visualisieren. Als Ausgangsbasis diente dabei eine frei verfügbare medizinische Beispieldatensammlung im Umfang von 1239 Einzeldokumenten in sechs Textdateien ([CFC]). Die Relevanzwerte R und ihre Bezeichnungen für das entsprechende Anzeigefeld des Applets werden je Suchbegriff und einzelnem Dokument wie folgt vergeben:

- Kommt der Suchbegriff im Dokumenttitel oder unter den Hauptthemen (*major subjects*) des Dokuments vor, so wird dieses als für diesen Suchbegriff vollständig relevant eingestuft (*full*, R = 1.0).

- Wurde der Suchbegriff in keinem der beiden obigen Felder gefunden, taucht aber unter den Nebenthemen (*minor subjects*) des Dokuments auf, so wird dieses als teilweise relevant betrachtet (*partial*, R = 0.5).

- Konnte der Suchbegriff lediglich im Dokumenttext selbst entdeckt werden, ohne ebenfalls in einem der bereits betrachteten Felder aufzutauchen, dann wird dies als wahrscheinlicher Zufallstreffer von geringer Relevanz bewertet (*incidental*, R = 0.25).

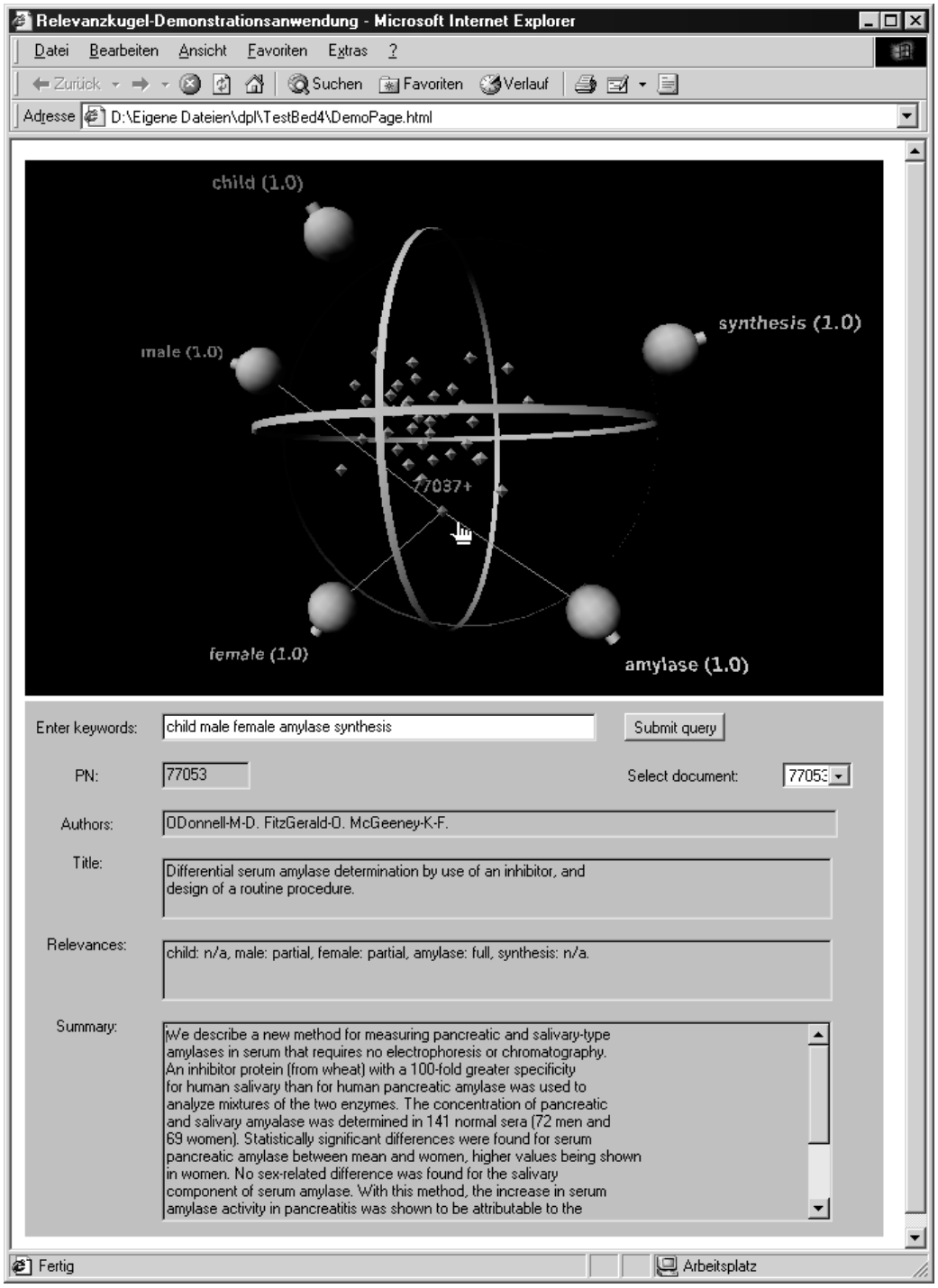

Abb. 20: VRML-Relevanzkugel und Testapplet

- Konnte der betreffende Suchbegriff in überhaupt keinem der bisherigen Dokumentfelder entdeckt werden, dann führt dies zu einer Einstufung des Dokuments als irrelevant für diesen Begriff (*n/a*, R = 0).

Dokumente mit gleichen Relevanzwerten zu allen Stichworten werden vom Applet in je einem Datenobjekt zusammengefaßt, welches sie alle gemeinsam repräsentiert und in der Darstellung durch ein auf die Kennummer[1] folgendes Pluszeichen im Label gekennzeichnet wird. Vollständig irrelevante Dokumente werden von vornherein nicht in die Darstellung aufgenommen. Da die Anzeige des bestehenden Applets immer nur den Inhalt eines Dokuments auf einmal darstellen kann, wurde zusätzlich die Möglichkeit eingerichtet, dieses über ein Pull-Down-Menü aus der Liste der in der Relevanzkugeldarstellung gerade markierten Dokumente auszuwählen.

Zur Bewertung der Implementierung ziehen wir zunächst noch einmal die einzelnen Punkte der Aufgabenstellung aus Abschnitt 4.1. (Anforderungsanalyse) heran:

1.) *Funktionalität.* Die hier besprochene Implementierung bietet dem Anwender auf jeden Fall die grundsätzlichen Funktionen des Relevanzkugelmodells: die freie Positionier- und innerhalb definierter Grenzen ebenfalls freie Gewichtbarkeit der Referenzobjekte ebenso wie eine Möglichkeit, einzelne Datenobjekte auszuwählen und gegebenenfalls eigens hervorzuheben. Das Fehlen des aus der Implementierung nach [Leissler 97] bekannten Relevanzkugelmanipulators macht sich soweit nicht negativ bemerkbar; im Rahmen der normalen Aufgabe des Modells erscheint eine explizit fest integrierte Möglichkeit, die Kugel über den Bildschirm zu ziehen, eher überflüssig, und die Rotation der Kugel durch Erfassen und Bewegen ihrer Ringelemente wirkt intuitiver als die Realisierung über ein räumlich von ihr getrenntes Zusatzteil. Die implementierten Methoden, bei Bedarf Zusatzinformationen zu bieten – Einblendung von Datenobjektlabels und Verbindungslinien sowie die farbliche Hervorhebung von Datenobjekten, für die das derzeit erfaßte Referenzobjekt relevant ist – haben sich prinzipiell bewährt; der Selektor zur simultanen Auswahl mehrerer benachbarter Datenobjekte ist hingegen eventuell noch verbesserungsfähig. (Die Interaktion zwischen kugelförmigem Auswahlbereich und oktaederförmigen Datenobjekten kann dazu führen, daß versehentlich mehr oder weniger Objekte aus- oder abgewählt werden als intuitiv erwartet.) Auf der Programmebene kann die Kugel sowohl mit anderen VRML-Knoten als auch über das External Authoring Interface mit externen

[1] Als Label wird jeweils die 'paper number' des einzelnen Dokuments bzw. für Mehrfachobjekte die des ersten gefundenen benutzt.

Applets oder Anwendungen Daten austauschen, so daß die Anbindung an die bildlich zu repräsentierende Informationsmenge gegeben ist; im vorgestellten Testfall gestaltete sich diese Kommunikation problemlos.

2.) *Einfache Bedienung.* Für den Anwender ist, einmaliges Verständnis für das Prinzip einmal vorausgesetzt, die Bedienung der Relevanzkugel mit dem allgemein verbreiteten Werkzeug 'Mauszeiger' praktisch trivial. Die Auswahl von Objekten durch Anklicken ebenso wie die weitere Manipulation durch Mausbewegungen bei heruntergehaltener Taste sind bei heute gebräuchlichen graphischen Benutzeroberflächen Standard, und die hier beschriebene Relevanzkugel benutzt VRML-Sensorknoten aus dem Standard-Sprachumfang, die genau diese Arten von Ereignissen erkennen und weitergeben können. Das Problem der Identifizierung einzelner Datenobjekte ist mit Hilfe der bei Bedarf einblendbaren Labels zufriedenstellend gelöst. – Auf der programmtechnischen Ebene sind insbesondere die Schnittstellen des Datenobjekt- und des ihn enthaltenen Relevanzkugelknotens etwas umfangreich ausgefallen. Dies liegt teilweise daran, daß die Implementierung aus technischen Gründen (siehe A.1.) keine VRML-*exposedFields* verwenden kann, die das Interface eventuell kompakter gestalten würden, und ist zum anderen darin begründet, daß die reine *Darstellung* zwecks Plattformunabhängigkeit möglichst flexibel sein soll und daher eine Menge von Einstellungsoptionen bietet.

3.) *Leistungsfähigkeit.* Auf diesem Gebiet sind die primären einschränkenden Faktoren einmal die Leistungsfähigkeit der Kombination aus VRML-Browser und zur Verfügung stehender Hardware an sich und zum anderen der zur Verwaltung der Szene und sämtlicher Ereignisse sowie zur Bestimmung der Datenobjektpositionen Aufwand. Davon läßt sich der erste Faktor von der Entwicklerseite aus nicht beeinflussen; welches System der Anwender genau benutzt, steht allein in dessen Ermessen. Der Rechenaufwand allerdings läßt sich minimieren, beispielsweise, indem man keine unnötigen Einzelknoten verwendet und möglichst wenige Ereignisse – insbesondere gleichzeitig – generiert und über Signalrouten weitergeleitet werden. Außerdem genügt es zur Positionsbestimmung eines Datenobjekts im allgemeinen, wenn nur jeweils die nötige Positions*änderung*, die sich ja direkt aus der Differenz von alten und neuen Positionsvektoren und Gewichtswerten des gerade manipulierten Referenzobjekts und dem individuellen Relevanzfaktor ableiten läßt,

berechnet und direkt zur vorherigen Position addiert wird. – Im Rahmen des Testszenarios war keine Einschränkung der Echtzeitreaktionsfähigkeit der Relevanzkugel festzustellen. Dies könnte prinzipiell an der zu geringen Anzahl von betroffenen Datenobjekten liegen, da die zu deren Positionsberechnung nötige Zeit zwangsläufig linear mit ihrer Zahl wächst. Jedoch ergab sich selbst bei einer eigens definierten separaten Beispielkugel mit sechs Referenz- und zweihundert Datenobjekten, für die jeweils alle Referenzobjekte mehr oder weniger relevant waren, auf dem verwendeten Rechner (ein schon etwas älterer Pentium III ohne besondere Graphikhardware) allenfalls bei genauem Hinsehen eine kaum merkliche Reaktionsverzögerung, die ebenso gut vom erhöhten Renderingaufwand für alle Einzelflächen wie von der Positionsberechnung selbst herrühren könnte; eine für das menschliche Auge merkbar verzögerte Reaktion auf Berührung oder Anklicken eines Datenobjekts konnte überhaupt nicht beobachtet werden.

4.) *Plattformunabhängigkeit.* In diesem Bereich besteht vermutlich die größte Schwäche dieser Realisierung. Zwar bietet VRML unbestritten eine plattformübergreifende standardisierte Implementierungsgrundlage, so daß die Relevanzkugel ohne weiteres auf allen standardkonformen VRML-Browsern lauffähig sein sollte; jedoch ist nicht notwendigerweise jeder Browser *tatsächlich* bis ins Letzte standardfest. (Der Cortona-Browser von Parallelgraphics kommt in seinen aktuellen Versionen mit der Relevanzkugel problemlos zurecht; dasselbe gilt jedoch nicht unbedingt für ältere Browsermodelle oder solche mit reduziertem Funktionsumfang.) Ein zweites Problem in diesem Zusammenhang besteht darin, daß VRML in der Praxis nicht die Art von Verbreitung gefunden hat, wie es für eine allgemeine Verwendung dieses Relevanzkugelmodells, etwa im Rahmen weltweit operierender Internet-Suchmaschinen, wünschenswert wäre. Beispielsweise sind gängige VRML-Browser oft ausschließlich als Plug-ins für Microsofts marktbeherrschenden Internet Explorer erhältlich, für Benutzer anderer Browsermodelle und Betriebssysteme stehen die Chancen, ein solches Plug-in für *ihren* Browser zu finden, das dann auch tatsächlich die volle VRML-Funktionalität samt EAI-Unterstützung bietet, dagegen eher schlecht. Damit wird aber eine Art von Plattformabhängigkeit gewissermaßen durch die Hintertür wieder eingeführt; von einer theoretisch denkbaren *universellen* Verwendbarkeit der VRML-Relevanzkugel kann damit nicht mehr die Rede sein. Auf der anderen Seite hat sich bislang neben

VRML kein anderer Standard für interaktive Web-basierte 3D-Graphikanwendungen wirklich etabliert. Es befindet sich zwar ein prinzipiell zu VRML rückwärtskompatibler offener Standard in Gestalt von *Extensible 3D* (X3D) in Entwicklung, diese ist aber zur Zeit noch nicht abgeschlossen.

5.) *Kompaktheit.* Diese Anforderung ist erfüllt. Die einzelnen Prototyp- und Skript-Bytecodedateien bewegen sich im Größenbereich von maximal ca. 20 Kilobyte; selbst zusammengenommen nehmen sie weniger Platz in Anspruch als viele Graphikdateien, wie man sie auf Webseiten häufig findet.

Wo nun genau die Grenzen der Darstellung liegen, ist schwer festzumachen. Beispielsweise gestaltete sich die oben unter Punkt 3.) erwähnte 200-Datenobjekt-Kugel unter Verwendung der normalen Voreinstellungen sehr unübersichtlich, da es unter den Datenobjekten zu vielen rein optischen Teilüberlappungen kam; eine einfache Halbierung der Datenobjektdimensionen in allen drei Richtungen verbesserte die Anschaulichkeit jedoch sofort. Allgemein läßt sich sagen, daß die Klarheit der Darstellung im konkreten Anwendungsfall in erster Linie von Anzahl und Größe der Datenobjekte, in zweiter von der Größe des verfügbaren Browserfensters beeinflußt wird. Je mehr Objekte in die Kugel gepackt werden sollen, um so kleiner muß jedes einzelne sein, wobei sich logischerweise eine Untergrenze durch die mögliche Bildschirmauflösung ergibt. Andererseits muß sich das Fenster, in dem die Relevanzkugel abgebildet wird, in der Regel den Bildschirm mit anderen Elementen teilen und wird daher normalerweise in beide Richtungen nicht wesentlich größer als einige hundert Bildpunkte ausfallen; die für die Testanwendung gewählte Größe von jeweils 640 * 400 Punkten sowohl für das Fenster des VRML-Browsers als auch für das Applet selbst ist für einige gängige Bildschirmeinstellungen bereits *sehr* großzügig gewählt. Es läßt sich daher grob abschätzen, daß die hier gezeigte Implementierung bis zu einige hundert Datenobjekte handhaben kann, solange diese klein genug definiert und nicht zu viele Labels zur gleichen Zeit eingeblendet sind; es könnte in diesem Fall sogar empfehlenswert sein, die Option zur Labeleinblendung für markierte Datenobjekte überhaupt abzuschalten. Spätestens bei der nächsthöheren Größenordnung von einigen *tausend* Datenobjekten ist sowohl fest mit Darstellungs- als auch, da natürlich der Rechen- und Renderingaufwand mit der Objektzahl steigt, wahrscheinlich mit Performanzproblemen zu rechnen. In Bezug auf die *Referenz*objekte dürfte die praktische Obergrenze bei ungefähr zehn liegen, da einerseits wie in 2.3. ausgeführt die Darstellung mit zunehmender Anzahl der zur Positionierung der Datenobjekte beitragenden Faktoren zunehmend Mehrdeutig-

keiten aufweist und andererseits der menschliche Betrachter selbst nur mit einer begrenzten Anzahl von relevanten Begriffen zur selben Zeit sinnvoll arbeiten und die Ergebnisse dieser Arbeit auch auswerten kann.

6.3. Gedanken zu Verbesserungsmöglichkeiten

Es ist nicht unbedingt einfach, unmittelbar bei Abschluß einer Implementierung sofort wieder Ideen zu entwickeln, wie diese weiter verbessert werden könnte. Hinzu kommt, daß die Relevanzkugel von ihrer Grundidee her ein eher einfaches Werkzeug ist und nach Möglichkeit auch bleiben soll, um ihren Nutzen nicht durch Überladen mit diversen Extrafunktionen am Ende möglicherweise zu gefährden. Nichtsdestotrotz ergeben sich einige vorstellbare Optionen.

- Eine denkbare Verbesserung könnte beispielsweise im Bereitstellen einer oder mehrerer Möglichkeiten bestehen, überflüssige Objekte aus der Darstellung zu entfernen, um diese übersichtlicher zu gestalten. Dies ist tatsächlich im Prinzip mit der existierenden Implementierung bereits realisierbar, indem statt der Relevanzkugel selbst *die externe Anwendung* geeignete Filterfunktionen anbietet und mit der Relevanzkugelkomponente zusammenarbeitet, um deren Darstellung entsprechend anzupassen.

- Alternativ oder zusätzlich wäre eine Zoomfunktion denkbar, die insbesondere bei eng gepackten Datenobjekten eine willkommene Ausschnittvergrößerung bewirken würde. Prinzipiell können zwar die Navigationsmöglichkeiten gängiger VRML-Browsermodelle herangezogen werden, um einen ähnlichen Effekt zu erzielen, indem sich der Betrachter selbst durch die Kugeldarstellung bewegt; allerdings geht in diesem Fall schnell der Überblick und insbesondere die Sicht auf die Referenzobjekte verloren.

- Es wäre auch möglich, den in dieser Implementierung nicht weiter verfolgten Ansatz zur Modellierung verformbarer Datenobjekte wieder aufzunehmen und auf seine konkrete Praxistauglichkeit zu prüfen. Für echte deformierbare Objekte mag sich dabei eine andere Plattform als VRML geeigneter erweisen; eine Darstellung als Kombination mehrerer beweglicher Teile erscheint dagegen auch mit deren Mitteln unter Verwendung zusätzlicher Knoten prinzipiell realisierbar.

- Verbesserungen an den Details und Schnittstellen der einzelnen Knotenprototypen sind immer denkbar. Zum Beispiel werden in dieser Implementierung dieselben Farben für alle Referenzobjekte gleichermaßen beziehungsweise sowohl für die Datenobjekte selbst als auch deren Labels und eventuelle Verbindungslinien verwendet. Während dies ein Maß an Einheitlichkeit schafft und im letzteren Fall der Assoziation zwischen verschiedenen Teilen desselben Konzepts dienlich ist, könnte es sich trotzdem als sinnvoll erweisen, zusätzliche Optionen etwa für Sonderfälle zur Verfügung zu stellen.

- Eine echte Verbesserung ließe sich eventuell erzielen, wenn die internen Abläufe der Kugel konsequent auf Multithreading umgestellt würden. Zur Erinnerung: Nach [VRML 97] unterscheidet VRML bei der Ereignisbehandlung in Java-Skripten zwischen Haupt- und Nebenthreads. Durch Auslagern von kritischen Ausgabeereignissen in solche Nebenthreads sollte prinzipiell das in Abschnitt 5.2. beschriebene bei Ausgaben im Hauptthread auftretende Gleichzeitigkeitsproblem umgangen werden können; bei geeigneter Implementierung könnte vielleicht sogar die Notwendigkeit für die derzeit nötigen Update-Signale entfallen und damit sowohl die Konstruktion an sich als auch die Schnittstelle nach außen deutlich eleganter gestaltet werden. Im Rahmen dieser Implementierung wurde dieser Ansatz nicht weiterverfolgt, da sich im Experiment nicht sicher bestätigen ließ, daß die verwendeten VRML-Browser den hier kritischen Unterschied zwischen Haupt- und Nebenthreads tatsächlich korrekt handhaben; auch müßten die derzeit bestehenden Knotenskripte zu diesem Zweck praktisch vollständig neu geschrieben werden.

6.4. Zukünftige Perspektiven

Der grundlegende Relevanzkugel*ansatz* mag seine Probleme haben und neben modernen virtuellen Realitätsumgebungen und Echtzeitsimulationen fast schon primitiv wirken, aber gerade seine Einfachheit dürfte auch in Zukunft seine Eignung als Werkzeug zur schnellen Vorselektion von Informationsinhalten einigermaßen sicherstellen. Daneben zeigt die vorliegende Implementierung durchaus Möglichkeiten auf, die inhärenten Schwächen des Modells weitestgehend zu kompensieren. Die interessantere Frage ist, auf welcher *Plattform* zukünftige Realisierungen dieses Konzepts aufbauen könnten. Für proprietäre Informationssysteme könnten natürlich prinzipiell jeweils ihre eigenen Versionen als Visualisierungshilfen entwickelt werden; eine eventuelle real existierende Monopolstellung eines bestimmten Anbie-

ters einmal ausgenommen, ist auf dieser Basis eine allgemeine Verbreitung eher nicht zu erwarten. Das World Wide Web bietet vermutlich auch in naher bis mittelfristiger Zukunft die beste Grundlage für eine solche: ein Alternativsystem, welches ähnlich universellen Zugriff auf ungeheure Informationsmengen bietet, ist schlichtweg nicht in Sicht, Web-Browserprogramme sind überall zu finden und im allgemeinen sogar frei verfügbar, und gerade die im Web vertretenen Internet-Suchmaschinen könnten von Methoden zur besseren Veranschaulichung ihrer Ergebnisse nur profitieren. Zur Zeit scheint es jedoch trotz der außerhalb von Fachkreisen eher beschränkten Verbreitung von VRML auf dem Gebiet Web-basierter dreidimensionaler Grafiken mit Interaktionsmöglichkeiten keine echte, geschweige denn beliebtere Alternative zu dieser Sprache zu geben. Dies mag sich mit der offiziellen Einführung von X3D ändern, sobald der Standard fixiert ist und entsprechende Werkzeuge auf den Markt kommen; in diesem Fall könnte es sich durchaus anbieten, über eine Neuimplementierung des Relevanzkugelkonzepts auf dieser Grundlage nachzudenken.

Anhang A: Spezifikation der VRML-Prototypen

A.1. Vorbemerkungen

Dieser Anhang beschreibt die einzelnen im Rahmen der Implementierung erstellten VRML-Knotenprototypen. Die Art der Beschreibung lehnt sich dabei an Kapitel 6 (*Node reference*) des Standarddokuments [VRML 97] an, d.h., sie besteht für jeden Prototyp aus seiner formalen Schnittstellendeklaration gefolgt von individuellen Anmerkungen zu den einzelnen Datenfeldern sowie Signalein- und -ausgängen. Die Länge der Deklarationen insbesondere des Relevanzkugel- und Datenobjektprototyps ist dabei nur teilweise der Komplexität der betreffenden Knoten geschuldet; bei genauerer Betrachtung finden sich bei diesen Knoten viele Felder *XXX*, denen ein gleichnamiger Eingang *set_XXX* und Ausgang *XXX_changed* zugeordnet sind. Die Verwendung von theoretisch analogen exposedFields zur Verkürzung der Deklaration verbietet sich jedoch aus zwei Gründen. Zum einen sind diese Felder, Ein- und Ausgänge an interne Skriptknoten gekoppelt, zu deren Definition aber explizit keine exposedFields verwendet werden dürfen ([VRML 97], 6.40, *Script*) und die auch in der Prototypendefinition nicht an solche angebunden werden können ([VRML 97], 4.8.3, *PROTO definition semantics*). Zum anderen werden über die entsprechenden Eingänge eintreffende Ereignisse, die generell Informationen zur nächsten beabsichtigten Anpassung der Kugeldarstellung transportieren, in der Praxis erst einmal zwischengespeichert und erst nach Eingang eines Update-Signals tatsächlich übernommen und an die zugehörigen Ausgänge weitergereicht; in in der Beschreibung der einzelnen Prototypen jeweils angegebenen Einzelfällen dient der entsprechende Ausgang obendrein zusätzlich zur Ausgabe von kugelintern generierten Ereignissen, so daß es zumindest theoretisch – bei entsprechender Verzögerung des Updates nach einer externen Eingabe, während der Benutzer *gleichzeitig* weiter mit der bestehenden Relevanzkugel arbeitet – denkbar ist, daß zwischen Signaleingang und Reaktion darauf eine oder mehrere Ausgaben erfolgen, die zu keinem von beiden in Beziehung stehen. Da dieses Verhalten stark vom normalen VRML-Ereignismodell abweicht, das alle Ereignisse einer Kaskade als prinzipiell gleichzeitig betrachtet ([VRML 97], 4.10.3, *Execution Model*), wäre eine Abkürzung vermittels exposedFields nicht einmal dann sinnvoll, wenn sie möglich wäre, da diese sich dann naturgemäß in dieser Beziehung von allen anderen exposedFields, wie sie normalerweise verwendet werden, unterscheiden würden.

Soweit nicht im Einzelfall anders beschrieben, werden solche in der Deklaration auftretenden Tripletts von EventIn/Feld/EventOut im Rahmen der folgenden Beschreibung unter ihrem Feldnamen zusammengefaßt und folgen dem oben erläuterten Ablaufschema, indem eingehende Ereignisse zunächst zwischengespeichert (wobei gegebenenfalls ein neueres eintreffendes Ereignis ein altes überschreibt) und dann nach Eintreffen eines Update-Signals alle gemeinsam ausgewertet werden; erst im Rahmen dieses Auswertungsprozesses generieren die zugeordneten EventOuts die entsprechenden ausgehenden Ereignisse.

A.2. Prototyp ReferenceObject

```
ReferenceObject {
  exposedField SFNode spherePartMaterial Material {}
  exposedField SFNode buttonPartMaterial Material {}
  exposedField SFNode labelMaterial Material {}
  field SFInt32 index -1
  field SFString label ""
  field SFBool showWeight TRUE
  field SFVec3f mainTranslation 0 3 0
  field SFVec3f offsetTranslation 0 0 0
  field SFRotation rotation 0 1 0 0
  field SFFloat weight 0
  field SFVec3f objectScale 0.2 0.2 0.2
  eventOut SFRotation rotation_changed
  eventOut SFFloat weight_changed
  eventOut SFVec3f position_changed
  eventOut SFVec3f endPoint_changed
  eventOut SFInt32 objectStats_changed
  eventOut SFInt32 touchedObject_changed
  eventOut SFInt32 grabbedObject_changed
}
```

Daß der Referenzobjekt-Knotenprototyp praktisch keine Signaleingänge aufweist (außer den mit den drei zur Aufnahme der Materialknoten für seine einzelnen Komponenten bestimmten exposedFields assoziierten), liegt darin begründet, daß ein Referenzobjekt in der hier implementierten Fassung Zeit seiner Existenz keine eingehenden Ereignisse empfangen und verarbeiten muß; in Fällen, die dies theoretisch notwendig machen könnten, ersetzt das Relevanzkugelskript einfach die vorhandenen Referenzobjekte durch neue. Es genügen daher Datenfelder zur Aufnahme von Startwerten und Signalausgänge, über die das Referenzobjekt – primär durch Manipulationen des Betrachters verursachte – Veränderungen signalisieren kann.

Die Felder *spherePartMaterial*, *buttonPartMaterial*, und *labelMaterial* enthalten jeweils einen Materialknoten, der die für solche Knoten typischen Eigenschaften – Farbe, Transparenz, und Reflexionsverhalten – des entsprechenden Bestandteils

des Referenzobjekts definiert, also *spherePartMaterial* für die Basiskugel, *button-PartMaterial* für den Bedienungsknopf, und *labelMaterial* für den Schriftzug.

index gibt die relevanzkugelinterne Nummer des Referenzobjekts an und ist eigentlich nichts anderes als die Position, an der die das Objekt beschreibenden Werte in den relevanten Listen gefunden werden können.

label spezifiziert die Zeichenkette, die dem dargestellten Referenzobjekt zur Identifizierung mitgegeben werden soll; gleichzeitig gibt *showWeight* an, ob an diese Zeichenkette in der Darstellung eine explizite (und aktuell gehaltene) numerische Gewichtsanzeige angehängt werden soll oder nicht.

mainTranslation und *offsetTranslation* definieren den Abstand des Referenzobjekts vom Relevanzkugelzentrum. Hierbei repräsentiert *mainTranslation* den Relevanzkugelradius an sich, *offsetTranslation* dagegen einen optionalen Zusatzabstand von eher kosmetischer Bedeutung, der gegebenfalls das Referenzobjekt ein wenig mehr von der Relevanzkugelschale abheben kann, um Teilkollisionen mit in der Nähe positionierten Datenobjekten zu vermeiden. In beiden Fällen ist 'Abstand' definiert als die Distanz zwischen dem Relevanzkugelzentrum und dem Punkt des Referenzobjekts, der diesem am nächsten kommt, so daß bei einem Zusatzabstand von 0 das Referenzobjekt gerade auf der Relevanzkugelschale aufliegt. Obwohl beide Felder strenggenommen beliebige dreidimensionale Vektoren enthalten können, werden im Rahmen der Implementierung nur die y-Komponenten verwendet und betrachtet.

rotation und *weight* sind die eigentlichen Kenngrößen des Referenzobjekts. *rotation* gibt seine Startposition auf der Kugelschale in Form eines VRML-Rotationswerts an, der die Drehung beschreibt, die aus der gedachten Ausgangsposition am Nordpol der Relevanzkugel um deren Zentrum herum ausgeführt werden müßte, um das Referenzobjekt an seine gegenwärtige Position zu bringen. *weight* hingegen spezifiziert sein Startgewicht zu Beginn. Beide Werte können sich natürlich im Laufe der Zeit, die das Referenzobjekt existiert, vielfach ändern.

objectScale gibt einen globalen Skalierungsfaktor für das Referenzobjekt vor, der auf alle Komponenten gleichermaßen angewendet wird und damit prinzipiell deren Größe festlegt. Der voreingestellte Defaultwert beruht auf Erfahrung; bei einem hypothetischen *objectScale*-Wert von 1 1 1 betrüge der maximal mögliche Radius

des Kugelteils eines Referenzobjekts, d.h. bei maximalem Gewicht, volle zwei Drittel (!) des normalerweise voreingestellten Relevanzkugelradius selbst.

rotation_changed, *weight_changed*, *position_changed* und *endPoint_changed* dienen jeweils zur Ausgabe von Detailinformationen über den Status des Referenzobjekts, während dieses manipuliert wird. Die beiden erstgenannten liefern dabei natürlich aktuelle Informationen über Rotationswert respektive Gewicht. *position_changed* hingegen gibt die eigentliche Positionsinformation aus, die zur Berechnung der Datenobjektpositionen im Inneren der Relevanzkugel herangezogen ist; dabei erfolgen die Projektion der rein physischen Position des Referenzobjekts auf die Relevanzkugelschale sowie die Umrechnung in kartesische Koordinaten bereits referenzobjektintern, so daß der ausgegebene Wert direkt weiterverwendet werden kann. *endPoint_changed* schließlich gibt die aktuelle Position des Mittelpunkts des Referenzobjektkugelteils an; der etwas eigenwillige Name erklärt sich daraus, daß eventuell gezogene Verbindungslinien zwischen Referenz- und Datenobjekt gerade an diesem Punkt festgemacht werden.

Ironischerweise ist keiner der vier obigen Signalausgänge direkt über Routen mit dem Datenobjektknoten verbunden. Zu diesem Zweck existiert *objectStats_changed*, das bei jeder Änderung an einem oder mehreren dieser Ausgänge die Indexnummer des Referenzobjekts ausgibt und so signalisiert, daß neue Werte abfragebereit zur Verfügung stehen. Das eigentliche Auslesen der Werte übernimmt der Datenobjektknoten in eigener Initiative.

touchedObject_changed und *grabbedObject_changed* schließlich reagieren weitgehend analog zu *objectStats_changed* mit der Ausgabe der Referenzobjektnummer, sobald das Objekt vom Betrachter mit dem Mauszeiger berührt bzw. erfaßt wird. Endet die Berührung, oder wird das Objekt wieder losgelassen, so erfolgt eine zweite Reaktion, indem der betreffende Ausgang den Wert -1 annimmt, der aufgrund der Vergabepraxis der Indexnummern zu keinem Referenzobjekt gehören kann.

A.3. Prototyp DataObjects

```
DataObjects {
  eventIn SFInt32 set_changeAtRefObject
  eventIn SFNode set_color
  eventIn SFInt32 set_grabbedRefObject
  eventIn MFString set_labels
  eventIn SFFloat set_labelOffset
  eventIn SFVec3f set_labelScale
```

```
  eventIn MFInt32 set_marked
  eventIn SFNode set_material
  eventIn SFVec3f set_objectSize
  eventIn MFNode set_referenceObjects
  eventIn MFFloat set_relevances
  eventIn SFBool set_repaint
  eventIn SFNode set_selectorMaterial
  eventIn SFBool set_showLabelsOnTouch
  eventIn SFBool set_showLinesOnTouch
  eventIn SFBool set_showLabelsWhenMarked
  field SFNode color NULL
  field MFString labels []
  field SFFloat labelOffset 0
  field SFVec3f labelScale 0 0 0
  field MFInt32 marked []
  field SFNode material NULL
  field SFVec3f objectSize 0 0 0
  field MFFloat relevances []
  field SFNode selectorMaterial NULL
  field SFBool showLabelsOnTouch TRUE
  field SFBool showLinesOnTouch TRUE
  field SFBool showLabelsWhenMarked TRUE
  eventOut SFInt32 clickedObject_changed
  eventOut SFNode color_changed
  eventOut SFBool isRepainted
  eventOut MFString labels_changed
  eventOut SFFloat labelOffset_changed
  eventOut SFVec3f labelScale_changed
  eventOut MFInt32 marked_changed
  eventOut SFNode material_changed
  eventOut SFVec3f objectSize_changed
  eventOut MFFloat relevances_changed
  eventOut SFNode selectorMaterial_changed
  eventOut SFBool showLabelsOnTouch_changed
  eventOut SFBool showLinesOnTouch_changed
  eventOut SFBool showLabelsWhenMarked_changed
  eventOut SFInt32 touchedObject_changed
}
```

Der Datenobjekt-Knotenprototyp gestaltet sich ungleich komplexer als der vorige, da er im Gegensatz zu diesem ständig fester Bestandteil der Relevanzkugel ist und also auch entsprechende Ansteuerungsmöglichkeiten anbieten muß. Nichtsdestoweniger befassen sich auch hier die weitaus meisten Felder, Ein- und Ausgänge primär mit Details der reinen Darstellung; außerdem enthält die Deklaration einige der in Abschnitt A.1. beschriebenen Eingang/Feld/Ausgang-Tripletts, wodurch sie naturgemäß länger ausfällt.

set_changeAtRefObject ist der Signaleingang, über den die an den *objectStats_changed*-Ausgängen der Referenzobjektknoten am Datenobjektknoten eintreffen und ihn so über Änderungen der Positions- und Gewichtsverteilung der Referenzobjekte auf dem Laufenden halten.

color nimmt einen `Color`-Knoten auf, der die Farben angibt, die die einzelnen Datenobjekte je nach Zustand annehmen können. Der Normalfall sind vier Farben: in fest vorgegebener Reihenfolge je eine für unmarkierte bzw. markierte Datenobjekte allgemein gefolgt von denen für unmarkierte bzw. markierte Datenobjekte, die gerade mit dem Mauszeiger berührt worden sind. Listet der Color-Knoten weniger Farben auf, so werden die fehlenden Fälle nicht gesondert hervorgehoben (bei drei Farben beispielsweise erhält ein berührtes Objekt immer die dritte, unabhängig von seinem Markierungsstatus); enthält er mehr, werden die überzähligen Farben ignoriert.

Über den *set_grabbedRefObject*-Eingang erhält der Datenobjektknoten Rückmeldung darüber, ob ein Referenzobjekt gerade vom Betrachter erfaßt wurde, und um welches es sich handelt. Diese Information wird dann benutzt, um die einzelnen Datenobjekte, für die dieses Referenzobjekt relevant ist, so farblich hervorzuheben, als wären sie individuell berührt worden.

labels nimmt die Liste der Zeichenketten auf, mit denen die Datenobjekte bei entsprechender Einstellung in der Darstellung markiert werden können. Darüber hinaus ist es eines von zwei Feldern, die tatsächlich über die *Anzahl* der darzustellenden Datenobjekte entscheiden; das interne Skript geht von einem Datenobjekt pro Label oder pro angefangenem Satz von Relevanzen aus, je nachdem, welche Zahl größer ist, und füllt gegebenenfalls fehlende Daten in der anderen Liste mit Nullwerten auf. *labelOffset* und *labelScale* definieren den Abstand jedes Labels von der oberen Spitze seines oktaederförmigen Datenobjekts beziehungsweise seine Größe.

marked enthält zu jeder gegebenen Zeit die Information darüber, welche Datenobjekte gerade markiert sind und welche nicht. Da VRML keinen Listentyp für Wahrheitswerte anbietet, kommt statt dessen eine Liste von ganzen Zahlen zum Einsatz, wobei ein Wert von 0 angibt, daß das betreffende Datenobjekt unmarkiert ist, während markierte Objekte Werte ungleich 0 – klassischerweise -1 – erhalten. Während es sich bei *set_marked*/*marked*/*marked_changed* einerseits um eines der bewußten Tripletts handelt, generiert *marked_changed* notwendigerweise auch immer dann ein neues Ereignis, wenn der *Betrachter* ein Datenobjekt markiert beziehungsweise eine solche Markierung aufhebt.

material und *objectSize* definieren nach *color* die restlichen Darstellungseigenschaften der eigentlichen Datenobjekte. Während *color* die Oberflächenfarben selbst

spezifiziert, übernehmen die Datenobjekte die restlichen Eigenschaften, insbesondere Transparenz und Reflexionsverhalten, von dem von *material* aufgenommenen `Material`-Knoten wie in [VRML 97], Abschnitt 6.23 für `IndexedFaceSet`-Knoten definiert. Farben und Materialeigenschaften werden auch direkt für eventuell einzublendende Labels und/oder Verbindungslinien übernommen und richten sich nach dem jeweiligen individuellen Datenobjekt. *objectSize* definiert hingegen schlicht die maximalen Dimensionen der Datenobjekte in x-, y-, und z-Richtung.

set_referenceObjects erhält jedesmal, wenn vom Relevanzkugelskript neue Referenzobjektknoten generiert werden, eine Kopie der entsprechenden Liste; damit hat seinerseits das Datenobjektskript die Möglichkeit, nach Bedarf direkt auf die Signalausgänge dieser Knoten zuzugreifen.

relevances ist, wie sich aus dem Relevanzkugelkonzept leicht herleiten läßt, das zentrale Datenfeld des Datenobjektknotens schlechthin. Es enthält die Relevanzmatrix für alle aktuellen Referenz- und Datenobjekte – in Listenform, da VRML keine Arraytypen bereithält. Dabei setzt sich die Liste quasi aus den Relevanzlisten der einzelnen Datenobjekte in Folge zusammen, wobei für jedes Datenobjekt die Abfolge der betrachteten Referenzobjekte durch die von *set_referenceObjects* erhaltene Knotenliste strikt vorgegeben ist und auch Nullwerte nicht vernachlässigt werden dürfen.

set_repaint ist zuständig für den Empfang des Update-Signals vom Relevanzkugelskript (eingehender Wert TRUE), welches die Übernahme eventuell zwischengespeicherter Werte zur Anpassung der Darstellung veranlaßt. Umgekehrt sendet *isRepainted* die entsprechende Bestätigungsmeldung zurück, dies allerdings gegebenenfalls erst nach Empfang aller Referenzobjekt-Initialisierungsbestätigungen über *set_changeAtRefObject*, wodurch sich eine kleine Verzögerung ergeben kann.

selectorMaterial nimmt einen Material-Knoten zur Festlegung der optischen Eigenschaften – insbesondere natürlich der Transparenz – der Selektorkugel zur Simultanauswahl mehrerer Datenobjekte auf.

showLabelsOnTouch, *showLabelsWhenMarked*, und *showLinesOnTouch* sind einfache boolesche Felder, die festlegen, ob und wann Datenobjektlabels und Verbindungslinien zwischen Referenz- und Datenobjekten eingeblendet werden sollen. Wie bereits oben beschrieben, 'erben' diese zusätzlichen Darstellungselemente ihre Farb- und Materialeigenschaften von ihren jeweiligen Datenobjekten.

clickedObject_changed und *touchedObject_changed* schließlich signalisieren nach außen, welches Datenobjekt als letztes vom Beobachter angeklickt bzw. mit dem Mauszeiger berührt wurde. Dabei behält *clickedObject_changed* seinen Wert bis zum nächsten Mausklick über der Datenobjektgeometrie bei, während *touched-Object_changed* analog zu den entsprechenden eventOuts am Referenzobjekt-knoten ein Loslassen, das nicht direkt zum Kontakt mit einem *anderen* Datenobjekt führt, durch Ausgabe eines Werts von -1 signalisiert.

A.4. Prototyp Label

```
Label {
  exposedField SFNode fontStyle NULL
  exposedField SFNode material NULL
  exposedField SFVec3f scale 1 1 1
  exposedField MFString string ""
  exposedField SFVec3f translation 0 0 0
}
```

Hierbei handelt es sich um einen reinen Hilfsknotentyp, der zur Positionierung und Darstellung der Datenobjektlabel herangezogen wird; ein interner `Billboard`-Knoten sorgt dafür, daß die darzustellende Zeichenkette immer zum Betrachter hin orientiert wird, eine darüber hinausgehende Skriptfunktionalität ist jedoch nicht gegeben. Die einzelnen Felder haben die normalen Bedeutungen, wie sie von den in [VRML 97] beschriebenen Standardknotentypen her vertraut sind.

A.5. Prototyp RelevanceSphere

```
RelevanceSphere {
  eventIn SFFloat set_radius
  eventIn SFFloat set_offset
  eventIn SFRotation set_sphereRotation
  eventIn SFFloat set_ringWidth
  eventIn SFNode set_ringMaterial
  eventIn SFVec3f set_refObjectScale
  eventIn SFNode set_refSphereMaterial
  eventIn SFNode set_refButtonMaterial
  eventIn SFNode set_refLabelMaterial
  eventIn SFVec3f set_dataObjectSize
  eventIn SFNode set_dataObjectMaterial
  eventIn SFNode set_dataColor
  eventIn SFNode set_dataSelectorMaterial
  eventIn MFString set_refLabels
  eventIn MFRotation set_refRotations
  eventIn MFFloat set_refWeights
  eventIn SFBool set_showRefWeights
  eventIn MFString set_dataLabels
  eventIn SFVec3f set_dataLabelScale
  eventIn SFFloat set_dataLabelOffset
  eventIn SFBool set_showConnectionsOnTouch
  eventIn SFBool set_showDataLabelsOnTouch
  eventIn SFBool set_showDataLabelsWhenMarked
```

```
eventIn MFFloat set_dataRelevances
eventIn MFInt32 set_dataMarked
eventIn SFBool set_repaint
field SFFloat radius 3.0
field SFFloat offset 0.0
field SFRotation sphereRotation 0 0 1 0
field SFFloat ringWidth 0.1
field SFNode ringMaterial Material {}
field SFVec3f refObjectScale 0.2 0.2 0.2
field SFNode refSphereMaterial Material {}
field SFNode refButtonMaterial Material {}
field SFNode refLabelMaterial Material {}
field SFVec3f dataObjectSize 0.2 0.2 0.2
field SFNode dataObjectMaterial Material {}
field SFNode dataColor Color {color [0.8 0.8 0.8, 0 0.8 0, 1 1 1,
    0 1 0]}
field SFNode dataSelectorMaterial Material { transparency 0.7 }
field MFString refLabels []
field MFRotation refRotations []
field MFFloat refWeights []
field SFBool showRefWeights TRUE
field MFString dataLabels []
field SFVec3f dataLabelScale 0.3 0.3 0.3
field SFFloat dataLabelOffset 0.2
field SFBool showConnectionsOnTouch TRUE
field SFBool showDataLabelsOnTouch TRUE
field SFBool showDataLabelsWhenMarked TRUE
field MFFloat dataRelevances []
field MFInt32 dataMarked []
field SFBool autoStart TRUE
eventOut SFFloat radius_changed
eventOut SFFloat offset_changed
eventOut SFRotation sphereRotation_changed
eventOut SFFloat ringWidth_changed
eventOut SFNode ringMaterial_changed
eventOut SFVec3f refObjectScale_changed
eventOut SFNode refSphereMaterial_changed
eventOut SFNode refButtonMaterial_changed
eventOut SFNode refLabelMaterial_changed
eventOut SFVec3f dataObjectSize_changed
eventOut SFNode dataObjectMaterial_changed
eventOut SFNode dataColor_changed
eventOut SFNode dataSelectorMaterial_changed
eventOut MFString refLabels_changed
eventOut MFRotation refRotations_changed
eventOut MFFloat refWeights_changed
eventOut SFBool showRefWeights_changed
eventOut MFString dataLabels_changed
eventOut SFVec3f dataLabelScale_changed
eventOut SFFloat dataLabelOffset_changed
eventOut SFBool showConnectionsOnTouch_changed
eventOut SFBool showDataLabelsOnTouch_changed
eventOut SFBool showDataLabelsWhenMarked_changed
eventOut MFFloat dataRelevances_changed
eventOut MFInt32 dataMarked_changed
eventOut SFInt32 touchedDataObject_changed
eventOut SFInt32 clickedDataObject_changed
eventOut SFInt32 touchedRefObject_changed
eventOut SFInt32 grabbedRefObject_changed
eventOut SFBool isRepainted
```

```
}
```

radius und *offset* des Knotens legen den Radius der Relevanzkugel (und ihrer Ringelemente) beziehungsweise den optionalen Zusatzabstand der Referenzobjekte von der Relevanzkugelschale fest, definieren also bei der Erzeugung der einzelnen Referenzobjekte gerade deren *mainTranslation*- und *offsetTranslation*-Felder.

sphereRotation definiert die aktuellen Werte von Drehrichtung und -winkel der Kugel als Gesamtheit. Normalerweise wird wenig Veranlassung bestehen, dieses Feld von außen anzusteuern, wenn man nicht gerade mehrere Relevanzkugeln synchron in Drehung versetzen oder eine Darstellung aus einem bestimmten Blickwinkel sichern und später rekonstruieren will; nichtsdestotrotz generiert *sphereRotation_changed* bei vom Betrachter manuell vorgenommenen Drehungen entsprechende ausgehende Ereignisse zur Information.

ringWidth und *ringMaterial* beschreiben zusammen mit dem Relevanzkugelradius die Eigenschaften der drei kugeleigenen Ringelemente. Während der Radius offensichtlich den Durchmesser der Ringe festlegt, definiert *ringWidth* die Breite, und *ringMaterial* nimmt – wie prinzipiell alle im Rahmen dieser Prototypen auftretenden Felder, deren Namen auf *-Material* enden – einen `Material`-Knoten aus dem VRML-Standardrepertoire auf, der Farbe, Reflexionsverhalten, und Transparenz spezifiziert.

refObjectScale enthält direkt den einheitlichen Skalierungsfaktor für alle Referenzobjekte, wie in Abschnitt A.2. beschrieben. Analog nehmen *refSphereMaterial*, *refButtonMaterial*, und *refLabelMaterial* die Materialknoten zur Darstellung der einzelnen Referenzobjektkomponenten auf. Die hier gegebene Voreinstellung resultiert in einem neutralen undurchsichtigen Hellgrau entsprechend den Defaultwerten für `Material`-Knoten in [VRML 97], Abschnitt 6.27.

dataObjectSize, *dataObjectColor*, *dataObjectMaterial* und *dataSelectorMaterial* sind analog den *objectSize*-, *color*-, *material*- und *selectorMaterial*-Feldern des Datenobjektknotenprototyps; tatsächlich repräsentieren sie lediglich diese Felder des kugelinternen Datenobjektknotens nach außen, was ebenso für die jeweils assoziierten Signalein- und -ausgänge gilt.

refLabels, *refRotations*, und *refWeights* legen, respektive, die Bezeichner, Positionen auf der Kugelschale, und Gewichte der einzelnen Referenzobjekte fest.

Dabei bestimmt *refLabels* gleichzeitig die Anzahl der darzustellenden Objekte, nämlich je eins für jede in diesem Feld aufgeführte Zeichenkette; sollten eine oder beide der anderen Wertelisten kürzer sein, so werden für die verbleibenden Objekte Defaultwerte angenommen, während umgekehrt überzählige Werte ignoriert werden. *refRotations* definiert die Objektpositionen wie in Abschnitt A.2. beschrieben in Form von VRML-Rotationswerten; soweit für ein Referenzobjekt hier kein expliziter Wert angegeben wird, wird es an der Stelle positioniert, die es bei gleichmäßiger Anordnung aller Referenzobjekte in einem Kreis um die z-Achse herum (aus der normalen Anfangsposition des Betrachters gesehen entgegen dem Uhrzeigersinn) einnehmen würde. Für Referenzobjekte, die in *refWeights* kein explizites Gewicht zugeordnet erhalten, liegt der Fall dagegen einfacher: hier ist der angenommene Defaultwert schlicht 0. *refRotations_changed* und *refWeights_changed* sind, nebenbei, zwei weitere Signalausgänge, die bei Interaktionen des Betrachters mit der Kugel, hier speziell den Referenzobjekten, aktiv werden, um die aktuellen Werte nach außen hin verfügbar zu machen.

showRefWeights, *showConnectionsOnTouch*, *showDataLabelsOnTouch*, und *showDataLabelsWhenMarked* entsprechen genau den *showWeights*-, *showLinesOnTouch*-, *showLabelsOnTouch*-, und *showLabelsWhenMarked*-Feldern der Referenz- bzw. Datenobjektprototypen. Dabei gilt die *showRefWeights*-Einstellung einheitlich für alle Referenzobjekte.

Analog sind auch *dataLabels*, *dataLabelScale*, *dataLabelOffset, dataRelevances* und *dataMarked* ebenfalls nur Repräsentationen der entsprechenden Felder des Datenobjektknotens nach außen. Nicht so *autoStart*, das einzige Feld dieses Prototyps, mit dem keine Signalein- bzw. -ausgänge direkt verknüpft sind. Das liegt daran, daß dieses Feld nur während des Starts der Szene eine Funktion hat. Ist sein Inhalt TRUE, so wird zum frühestmöglichen Zeitpunkt ein automatisches Update gestartet, indem der Relevanzkugelknoten sein eigenes *set_repaint*-EventIn anspricht; dies führt zur Übernahme aller explizit mitgegebenen Startwerte (beziehungsweise, wo diese fehlen, Voreinstellungen) und zum sofortigen Aufbau einer Relevanzkugeldarstellung unter deren Verwendung. Steht *autoStart* dagegen auf FALSE, so geschieht so lange gar nichts, bis der Kugelknoten sein erstes Updatesignal von einer externen Quelle erhält. Dies erlaubt es, mit einer effektiv leeren Darstellung zu beginnen und diese erst einmal mit Werten und Eigenschaften zu versehen, bevor sie zum ersten Mal zur Anzeige kommt.

Es bleiben noch die Signalein- und -ausgänge zu betrachten, die nicht an eines der bisher genannten Felder geknüpft sind. Von zentraler Bedeutung sind zunächst *set_repaint* und *isRepainted*. *set_repaint* ist der Eingang für die vielbeschworenen Update-Signale; trifft dort ein Ereignis mit dem Wert TRUE ein, so interpretiert die Relevanzkugel dies derart, daß alle Eingaben von außerhalb abgeschlossen sind und in die Darstellung übernommen werden können. Umgekehrt signalisiert *isRepainted* den Abschluß des Updateprozesses, der je nach Leistungsfähigkeit der verwendeten Plattform und Komplexität der entstehenden Darstellung schon einmal einen wahrnehmbaren Moment dauern kann, an die Außenwelt.

Daneben existieren noch vier weitere Ausgänge, die *ausschließlich* Informationen über Interaktionen des Betrachters nach außen hin weitergeben. Diese sind bereits aus den vorhergehenden Prototypdeklarationen bekannt: *touchedDataObject_changed* und *touchedRefObject_changed* signalisieren, welches Daten- bzw. Referenzobjekt gerade in Kontakt mit dem Maus-zeiger ist, und nehmen den Wert -1 an, wenn der Mauszeiger das letzte berührte Objekt verläßt und in den leeren Raum zeigt. *lastClickedDataObject_changed* gibt an, welches Datenobjekt zuletzt durch Mausklick markiert bzw. in den Normalzustand zurückversetzt wurde, und *grabbedRefObject_changed* zeigt analog zu *touchedRefObject_changed* an, welches Referenzobjekt der Betrachter gerade erfaßt hat (und vermutlich manipuliert).

Anhang B: Quellenverzeichnis

B.1. Literatur

[Ames 97]: Andrea L. Ames, David R. Nadeau, John L. Moreland, *VRML 2.0 Sourcebook Second Edition*, John Wiley & Sons 1997

[Card 99]: Stuart K. Card, Jock D. Mackinlay, Ben Shneiderman, *Readings In Information Visualization: Using Vision To Think*, Morgan Kaufmann Publishers 1999

[Chen 99]: Chaomei Chen, *Information Visualization and Virtual Environments*, Springer-Verlag London 1999

[EAI]: *Information technology – Computer graphics and image processing – The Virtual Reality Modeling Language (VRML) – Part 2: External authoring interface (EAI)*, ISO/IEC 14772-2:2004

[Englberger 95]: Hermann Englberger, *Computergestützte Informationsvisualisierung – Eine Klassifikation aktueller Techniken und ihre Einsatzpotentiale für die Unternehmung*, Diplomarbeit November 1995, Technische Universität München, Fakultät für Informatik

[Hamilton 98]: Graham Hamilton, Rick Cattell, Maydene Fisher, *JDBC(TM): Datenbankzugriff mit Java(TM)*, Addison Wesley 1998

[Heilmann 96]: Marcus Heilmann, *Entwurf und Implementierung eines generischen Visualisierungswerkzeuges für gewichtete Relationen auf der Basis der direktmanipulativen Relevanzkugel-Metapher*, Diplomarbeit März 1996, Fachhochschule Darmstadt / GMD-Forschungszentrum Informationstechnik GmbH / Institut für integrierte Publikations- und Informationssysteme (IPSI)

[Hemmje 99]: Matthias Hemmje, *Unterstützung von Information-Retrieval-Dialogen mit Informationssystemen durch Informationsvisualisierung*, Dissertation, Darmstadt 1999

[Leissler 97]: Martin Leissler, Matthias Hemmje, *Portierung und Erweiterung der Relevanzkugelmetapher um interaktive Informationsvisualisierungsmechanismen*, GMD-Studien Nr. 321, GMD – Forschungszentrum Informationstechnik GmbH 1997

[Lemay 97]: Laura Lemay, Charles L. Perkins, *Java 1.1 in 21 Tagen*, SAMS 1997

[Lemay 98]: Laura Lemay, Rogers Cadenhead, *SAMS Teach Yourself Java 1.2 in 21 Days*, SAMS 1998

[Lemay 99] Laura Lemay, *Sams Teach Yourself Web Publishing with HTML 4 in 21 Days Professional Reference Edition*, Sams Publishing 1999

[Lohse 94]: Gerald L. Lohse, Kevin Biolsi, Neff Walker, Henry H. Rueter, *A Classification of Visual Representations*, Communications of the ACM, Vol. 37, No. 12, Dezember 1994

[Marrin 97]: Chris Marrin, *Proposal for a VRML 2.0 Informative Annex – External Authoring Interface Reference*, Silicon Graphics, 1997

[Müller/Schumann 00]: Heidrun Schumann, Wolfgang Müller, *Visualisierung*, Springer-Verlag 2000

[Roehl 97]: Bernie Roehl, Justin Couch, Cindy Reed-Ballreich, Tim Rohaly, Geoff Brown, *Late Night VRML 2.0 with Java*, Ziff-Davis Press 1997

[Shneiderman 96]: Ben Shneiderman, *The Eyes Have It: A Task by Data Type Taxonomy for Information Visualizations*, Proceedings of the 1996 IEEE Symposium on Visual Languages

[VRML 97]: *Information Technology – Computer graphics and image processing – The Virtual Reality Modeling Language (VRML) – Part 1: Functional specification and UTF-8 encoding*, ISO/IEC 14772-1:1997

[VRML A 02]: *Information Processing Systems – Computer Graphics – The Virtual Reality Modeling Language – Part 1 – Functional specification and UTF-encoding – Amendment 1 – Enhanced interoperability*, ISO/IEC 14772-1:1997/Amd. 1:2002(E)

[Watt 00]: Alan Watt, *3D Computer Graphics (Third Edition)*, Addison-Wesley 2000

B.2. Verwendete Software

Microsoft Windows ME

Microsoft Internet Explorer 5.5

ParallelGraphics Cortona VRML Client 4.2

Borland JBuilder Standard 2

Microsoft Word 2000

B.3. Testdaten

[CFC]: *Modern Information Retrieval – Cystic Fibrosis Reference Collection*, http://www.sims.berkeley.edu/~hearst/irbook/cfc.html

Anhang C: Abbildungsverzeichnis