Institut für Computervisualistik
Arbeitsgruppe Computergraphik

Studienarbeit

3D-Informationsvisualisierung in Information Retrieval Systemen. Recherche bestehender Ansätze und Implementierung einer Beispielapplikation.

von

Marc Pinto Santos

Betreuer und Prüfer: Prof. Dr. Stefan Müller

Februar 2004

Inhaltsverzeichnis

1. Einleitung 1
1.1. Motivation 1
1.2. Aufgabenstellung 1
1.3. Zusammenfassung 1
2. Informationsvisualisierung 3
2.1. Was ist Informationsvisualisierung? 3
2.2. Ziele der Informationsvisualisierung 4
3. Information Retrieval 6
3.1. Was leistet Information Retrieval? 6
4. Beispiele angewandter Informationsvisualisierung 9
4.1. Cone Tree 9
4.2. Collapsible Cylindrical Tree 10
4.3. Information Cube 11
4.4. Information Corridor 12
4.5. Perspective Wall 12
4.6. Documentfinder 13
4.7. Newsmap – Newsbrowser 14
4.8. MineSet 15
4.9. Polyarchie Visualisierung 15
4.10. Visualisierung von Geschäftsprozessen 16
4.11. Visualisierung der Entwicklung von WWW-Strukturen 17
5. Beispielanwendung: *INVENTORY 3D* 19
5.1. Ziele und Anforderungen 19
5.2. Architektur 21
5.2.1. Präsentationskomponenten 21
5.2.2. Szenenbeschreibung 21
5.2.3. Database Component 21
5.2.4. SceneManager 22
5.2.5. SceneGraph 22
5.2.6. DataGraph 22
5.2.7. Application Scripting and Messaging Engine (*INVENTORY 3D* - ASME) 22
5.2.8. Visualization-Wrapper & Viewer 22
5.2.9. Navigation 22
5.2.10. Konfiguration der Komponenten 22
5.2.11. Datenstrukturen 23
5.2.11.1. SceneGraph 23
5.2.11.2. DataGraph 25
5.2.11.2.1. Datenzugriff 25
5.2.11.2.2. Datenstruktur 26
5.2.11.2.3. Abbildungsstruktur 27
5.3. Implementierung 27
5.3.1. Verwendete Technologien 27
5.3.1.1. .NET-Framework von Microsoft 27
5.3.1.1.1. Microsoft Intermediate Language und Metadaten 27
5.3.1.1.2. Common Language Runtime 28
5.3.1.1.3. .NET Programmiersprachen und Interoperabilität 28
5.3.1.1.4. Plattformunabhängigkeit 29
5.3.1.1.5. Die C# Programmiersprache 29
5.3.1.1.6. Datenzugriff 29

5.3.1.1.7. Weitere .NET Features 30
5.3.1.1.8. Performanz 31
5.3.1.1.9. .NET als Grundlage für die Implementierung 31
5.3.1.1.10. .NET Speech 32
5.3.1.2. VRML Cortona Automation Interface (CAI) von Parallelgraphics 32
5.3.2. Bemerkungen zur Implementierung 32
5.3.2.1. Verwendete Tools 32
5.3.2.1.1. Microsoft Visual Studio .NET 2003 33
5.3.2.1.2. Openwave SDK 6.2.2 33
5.3.2.1.3. ParallelGraphics VrmlPad 2.0 33
5.3.2.1.4. NUnit 34
5.3.2.1.5. Microsoft Visio for Enterprise Architects 34
5.3.2.2. *INVENTORY 3D* Desktop Client 36
6. Zusammenfassung und Ausblick 39

Abbildungen und Tabellen

Abbildung 1: Referenzmodell des Ablaufs einer Informationsvisualisierung 5
Abbildung 2: Unterschiede zwischen Daten, Wissen und Informationen. 7
Abbildung 3: Ein Grundmodell für Information Retrieval 8
Abbildung 4: Visualisierung einer Hierarchie durch den Cone Tree. 9
Abbildung 5: Cam Tree 10
Abbildung 6: Collapsible Cylindrical Tree 10
Abbildung 7: Information Cube 11
Abbildung 8: Information Corridor 12
Abbildung 9: Perspective Wall 13
Abbildung 10: Document-Finder 13
Abbildung 11: Newsmap-Nachrichtenbrowser 14
Abbildung 12: MineSet von SGI 15
Abbildung 13 Transformation der Führungsstruktur zu Geschäftseinheiten 16
Abbildung 14 Multiple Hierarchien 16
Abbildung 15 Prozessgraph in 3D 17
Abbildung 16 Prozessgraph mit Histogramm 17
Abbildung 17 Time-Tube einer Web-Site über 4 Wochen 18
Abbildung 18: Architektur von *INVENTORY 3D* 19
Abbildung 19: Diagramm der Client/Server-Architektur 21
Abbildung 20: Visualisierungskomponenten des Desktop Clients 23
Abbildung 21: Schematische Darstellung des Szenegraphen 24
Abbildung 22: Schematische Darstellung des Datengraphs 25
Abbildung 23: Der Vrml-Editor VrmlPad im Einsatz 34
Abbildung 24: UML-Diagramm mit allen implementierten Szenegraphelementen 35
Abbildung 25: Darstellung des GUI des Desktop Clients 36
Abbildung 26: Bedienungspanel mit Analysefunktionen und Artikeldetails 37
Abbildung 27: Viewpoints von links nach rechts: Overview, Inside, Outside 38
Abbildung 28: Visualisierung nach Ausführen der Reorderfunktion 38

1. Einleitung

1.1. Motivation

Informationsvisualisierung ist eine relativ junge Informatikdisziplin, die alle Konzepte, Methoden und Anwendungen zur visuellen Darstellung von Informationen aus großen Dokumentsammlungen, Datenbanken oder anderen digitalen Bibliotheken zusammenfasst. Vor allem 3D-Visualisierung ermöglicht das Modellieren von sog. „Real World Metaphern", also 3D-Szenen, die einen Ausschnitt der realen Welt darstellen und es dem Benutzer ermöglichen, sich darin intuitiv zu bewegen.

1.2. Aufgabenstellung

Ziel dieser Arbeit ist es, bestehende Ansätze zur Informationsvisualisierung zu recherchieren, zu beschreiben und zu bewerten. Der Fokus liegt hierbei auf dreidimensionaler Darstellung von Informationen und Information Retrieval. Auf Basis dieser Recherche soll eine Beispielapplikation entworfen, geeignet umgesetzt und implementiert werden.

Schwerpunkte dieser Arbeit sind:

- Auflistung, Vorstellung und Bewertung moderner Informationsvisualisierungen in Information Retrieval Systemen.
- Entwurf einer Beispielapplikation, die im Rahmen einer realen Anwendung die entstehenden Daten dreidimensional visualisiert und dem Benutzer eine interaktive Bedienung ermöglicht.
- Implementierung der Beispielapplikation unter dem Einsatz eines modernen Frameworks. Als Programmiersprache soll hierbei die .NET Programmiersprache C# dienen. Weitere Features des .NET Frameworks, wie z.B. Speech.NET Beta 3 zur Sprachsteuerung, sollen in ihrer Funktionalität getestet werden und zum Einsatz kommen.

1.3. Zusammenfassung

Kapitel 2 ist als Einführung in die Informatikdisziplin „Informationsvisualisierung" gedacht.

Kapitel 3 befasst sich mit „Information Retrieval". Dabei wird der Begriff erläutert und in Zusammenhang gebracht mit Informationsvisualisierung.

In **Kapitel 4** werden Applikationen vorgestellt, die es dem Benutzer erlauben Daten dreidimensional zu visualisieren. Hierbei wird auf die Einsatzbereiche der jeweiligen Anwendungen eingegangen.

Kapitel 5 beinhaltet den Entwurf der Beispielanwendung *INVENTORY 3D*. Dabei werden Ziele und Anforderungen an die fertige Applikation erläutert. Ferner ist eine Übersicht über alle Bestandteile der Architektur gegeben, die um die Beschreibung der Komponenten und Datenstrukturen ergänzt wird. Erläuterungen zur Implementierung der Beispielapplikation finden sich im letzten Teil des Kapitels, wobei zuerst die verwendeten Technologien dargestellt werden. Daraufhin wird auf die Implementierung der einzelnen Komponenten der Anwendung eingegangen.

2. Informationsvisualisierung

2.1. Was ist Informationsvisualisierung?

Das Fachgebiet der Informationsvisualisierung ist relativ neu. Während die wissenschaftliche Visualisierung, etwa die grafische Repräsentation statistischer Daten oder Computeranimation von Prozessen in der Realwelt, allgemein bekannt und auch anerkannt ist, fristet die Informationsvisualisierung eher ein Schattendasein.

Um Informationsvisualisierung zu erklären, muss man auch auf Eigenschaften und Funktionen von Visualisierung eingehen. Visualisierung ist keine Erfindung der Moderne. Wenn man Visualisierung als einen Prozess versteht, bei dem Daten, Informationen und Wissen in visuelle Darstellungsformen transformiert werden, dann sind bereits die ersten Höhlenzeichnungen zu Beginn der menschlichen Kulturgeschichte eine Urform der visuellen Kommunikation. Die weitere Ausbildung dieser Kommunikationsform wurde auch dadurch unterstützt, dass der Mensch ein Lebewesen ist, das ca. 75% der Informationen aus visuellen Eindrücken der Realwelt rezipiert. Die restlichen 25% verteilen sich zur Hälfte auf auditive und sonstige Sinnesorgane. Man kann den heutigen Einsatz von Visualisierung in der computerisierten Welt als visuelles Kommunikationsmittel als Meilenstein einer Jahrtausende andauernden Entwicklung ansehen, oder als eine wichtige Schnittstelle im Man-Machine-Interface.

Es lassen sich vier elementare Aufgaben der Visualisierung formulieren[1]:

- Symbole, Diagramme oder Animationen helfen komplexe Prozessabläufe und Objektbeziehungen in der Realwelt zu veranschaulichen und gegebenenfalls zu vereinfachen.
- Visualisierung vereinfacht den Zugang zu Massendaten, z.B. durch Klassifikation und Datenstrukturierung.
- Visualisierung hilft bei der Analyse und Interpretation von Daten, bei der Sichtbarmachung verborgener Trends sowie bei der Mustererkennung.
- Visualisierung entspricht der Neigung der menschlichen Spezies und unserer Kultur, visuelle Informationsprozesse und Repräsentationsformen zu bevorzugen. Aus der Gehirnforschung ist darüber hinaus bekannt, dass sich Visualisierung positiv auf die Gedächtnisleistung und auf die menschliche Informationsaufnahme auswirkt.

Wie lässt sich Informationsvisualisierung definieren?

Unter Informationsvisualisierung versteht man heute eine Zusammenfassung aller Konzepte, Methoden und Tools zur visuellen Darstellung von Informationen aus Datenbanken, digitalen Bibliotheken oder anderen großen Dokumentsammlungen [2].

Wie bei vielen neuen Forschungsgebieten gibt es jedoch keine genaue Definition des Begriffs Informationsvisualisierung. Drei weitere Definitionsansätze sollen die Aufgaben und Ziele der Informationsvisualisierung noch etwas genauer beschreiben:

- Information Visualization is the use of computer-supported interactive visual representation of abstract data to amplify cognition [3].
- Information Visualization combines aspects of scientific visualization human-computer interfaces, data mining, imaging and graphics [4].
- Information visualization is the process of binding information that is not inherently spatial into a visual medium which the user can view and perceive [5].

Informationsvisualisierung beinhaltet also neben der visuellen Repräsentation abstrakter Informationen die computerunterstützte Aufbereitung derselben. Ziel jeder Variante von Informationsvisualisierung ist die Erleichterung des kognitiven Zugangs zu elektronisch gespeicherten Daten. Weiterhin hat die technologische Entwicklung im Bereich preiswerter High-Performance Computergrafik in den letzten Jahren zu großen Fortschritten in der wissenschaftlichen Datenvisualisierung geführt. Parallel dazu entwickelt sich die weltweite Vernetzung über das WWW[1] mit großer Geschwindigkeit. Dadurch werden immer mehr verteilte Informationsressourcen erschlossen und bereitgestellt, die zu einer kaum noch überschaubaren Menge an Informationen führen. Problematisch ist also das Überangebot an Informationen und somit das Auffinden, als auch das Verstehen derselben. Lösungsansätze, die genau dieses Problem angehen, findet man im Forschungsgebiet des Information Retrieval, worauf in einem folgenden Kapitel noch eingegangen wird. Eine weitere Fragestellung für Forschende in der Informationsvisualisierung war und ist, inwieweit Verfahren aus der wissenschaftlichen Visualisierung auf die Informationsvisualisierung übertragen werden können und was wissenschaftliche Messdaten von abstrakten Daten unterscheidet. Um diese Fragen zu beantworten, muss man sich mit den Zielen der Informationsvisualisierung befassen.

2.2. Ziele der Informationsvisualisierung

Primäres Ziel der Informationsvisualisierung ist es, abstrakte Informationen aus beliebigen, vorrangig textbasierten, Informationssystemen strukturell aufzubereiten und grafisch darzustellen. Dabei werden traditionelle Informationssysteme durch die visuelle Repräsentation von Informationen um folgende Aspekte erweitert [1]:

- Das Auffinden spezifischer Informationsdomänen in großen Datenbeständen.
- Das Erkennen von Relationen, Strukturen oder Trends in unstrukturiert erscheinenden Informationsmengen.
- Multiple Views, d.h. verschiedene Sichten auf identische Datenbestände.

Das folgende Referenzmodell nach [6] zeigt den klassischen Ablauf einer Informationsvisualisierung auf.

[1] Word Wide Web

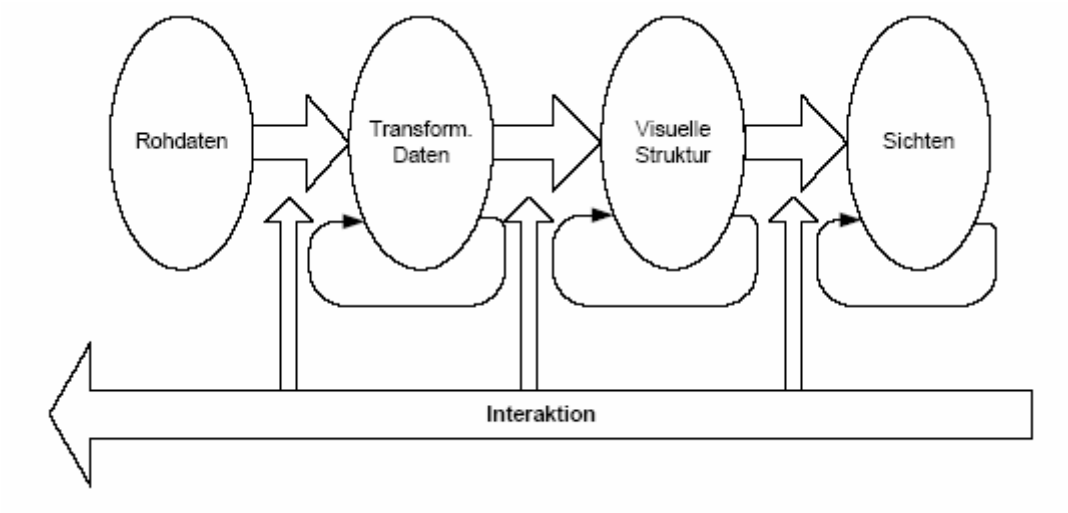

Abbildung 1: Referenzmodell des Ablaufs einer Informationsvisualisierung

Das Modell geht von einem Bestand von Rohdaten aus, welche in einer proprietären Struktur vorliegen. Das können z.B. relationale Tabellen oder XML sein. Es kann sich dabei aber auch um völlig unstrukturierte Daten handeln. Um von diesen Daten zu einer Visualisierung zu gelangen, müssen zunächst diverse Transformationen mit den Daten durchgeführt werden. Einerseits ist es möglich, dass abgeleitete Größen (z.B. Aggregationen) dargestellt werden sollen, andererseits müssen die Daten normalisiert werden, um eine Abbildung auf visuelle Eigenschaften möglich zu machen. Das Ergebnis dieses ersten Schrittes ist eine Menge transformierter Daten in einheitlicher Struktur. Im nächsten Schritt werden nun diese Daten auf die gewünschte visuelle Struktur abgebildet (Mapping). Auf dieser Struktur kann nun eine Menge von benutzerspezifischen Sichten generiert werden, die dem Benutzer die Navigation durch die Darstellung erlaubt. Eine Interaktion des Benutzers kann sich dann auf mehrere Stufen des Abbildungsprozesses auswirken. Er kann seine Sicht auf die Daten verändern, die visuelle Struktur ändern oder sogar auf die Datentransformation Einfluss nehmen. Die zyklischen Pfeile in der Abbildung verweisen auf die Tatsache, dass es sich hier um einen iterativen Prozess handelt, der zunächst mehrfach wiederholt werden kann, bevor der nächste Schritt durchgeführt wird.

3. Information Retrieval

3.1. Was leistet Information Retrieval?

Information Retrieval[2] beschäftigt sich mit der Informationssuche in kaum strukturierten Datenbeständen, die sich aus Texten und multimedialen Datenbasen zusammensetzen. Die wohl bekanntesten Anwendungsbeispiele für IR sind digitale Bibliotheken, Internet-Suchmaschinen und Bilddatenbanken.

Eine zeitgemäße und aktualisierte Definition von IR wurde von der Fachgruppe "Information Retrieval" innerhalb der "Gesellschaft für Informatik" aufgestellt:

"Im Information Retrieval werden Informationssysteme in Bezug auf ihre Rolle im Prozess des Wissenstransfers vom menschlichen Wissensproduzenten zum Informations-Nachfragenden betrachtet. Die Fachgruppe "Information Retrieval" in der Gesellschaft für Informatik beschäftigt sich dabei schwerpunktmäßig mit jenen Fragestellungen, die im Zusammenhang mit vagen Anfragen und unsicherem Wissen entstehen.

Vage Anfragen sind dadurch gekennzeichnet, dass die Antwort a priori nicht eindeutig definiert ist. Hierzu zählen neben Fragen mit unscharfen Kriterien insbesondere auch solche, die nur im Dialog iterativ durch Reformulierung (in Abhängigkeit von den bisherigen Systemantworten) beantwortet werden können; häufig müssen zudem mehrere Datenbasen zur Beantwortung einer einzelnen Anfrage durchsucht werden.

Die Darstellungsform des in einem IR-System gespeicherten Wissens ist im Prinzip nicht beschränkt (z.B. Texte, multimediale Dokumente, Fakten, Regeln, semantische Netze). Die Unsicherheit (oder die Unvollständigkeit) dieses Wissens resultiert meist aus der begrenzten Repräsentation von dessen Semantik (z.B. bei Texten oder multimedialen Dokumenten); darüber hinaus werden auch solche Anwendungen betrachtet, bei denen die gespeicherten Daten selbst unsicher oder unvollständig sind (wie z.B. bei vielen technisch-wissenschaftlichen Datensammlungen). Aus dieser Problematik ergibt sich die Notwendigkeit zur Bewertung der Qualität der Antworten eines Informationssystems, wobei in einem weiteren Sinne die Effektivität des Systems in Bezug auf die Unterstützung des Benutzers bei der Lösung seines Anwendungsproblems beurteilt werden sollte."

In der vorliegenden Beispielimplementation werden die Kriterien eines klassischen IR-Systems nicht erfüllt. Der Fokus während der Konzeption und Aufgabenstellung lag in einer geeigneten Umsetzung einer visuellen Metapher und einer effizienten und leicht verständlichen Benutzeroberfläche mit Spracheingabe. Die IR-Komponente wird in der Anwendung durch triviale Datenbankabfragen ersetzt.

[2] IR

Dennoch ist Informationsvisualisierung im Zusammenhang mit IR oder genauer IR-Systemen als feste Einheit zu sehen und zumindest theoretisch zu erarbeiten, da die durch ein IR-System ermittelten Daten trotz Aufbereitung nur schwer zu rezipieren sind. Hierbei kommt die Informationsvisualisierung ins Spiel, indem geeignete 2-dimensionale oder 3-dimensionale Visualisierungselemente als Träger für die unterschiedlichen Datentypen eingesetzt werden. Zu visualisierende Daten lassen sich in folgende Kategorien einteilen [7]:

1. Eindimensionale Daten: Hierbei handelt es sich um sequentiell organisierte Daten wie Textdokumente, Quellcode, Namenslisten etc. Jedes Informationsobjekt ist eine Textzeile bzw. ein Charakterstring.
2. Zweidimensionale Daten (Map-Data): Zweidimensionale Daten sind über einer Fläche gegebene Daten, wie z.B. Daten über einer geographischen Karte oder Zeitungslayout. Jedes Informationsobjekt ist einem bestimmten Bereich der Fläche zugeordnet.
3. Dreidimensionale Daten (World Data): Dazu zählen Objekte der realen Welt wie Häuser oder Moleküle, wobei diese Objekte untereinander in teilweise sehr komplexen Beziehungen stehen können.
4. Temporal Data: Zeitreihendaten sind Daten, bei denen die Informationsobjekte durch Zeitstempel gekennzeichnet sind und sich überlappen können.
5. Multidimensional Data: Hier besitzen die Informationsobjekte mindestens 2 Attribute. Bekanntestes Beispiel sind Daten in relationalen Datenbanken.
6. Tree-Data: Daten, die in einer Baumstruktur angeordnet sind, nennt man Tree-Data. Hierbei sind die Informationsobjekte hierarchisch strukturiert.
7. Network-Data: Network-Data bestehen aus netzwerkartig verknüpften Informationsobjekten.

Wie bereits erwähnt, wurde in der Beispielapplikation statt eines IR-Systems ein Datenbanksystem verwendet. Der grundlegende Unterschied liegt darin, dass Datenbanksysteme Daten enthalten und diese bereitstellen, während IR-Systeme den Benutzer bei der Suche nach Informationen unterstützen und dabei auf sog. Wissensbasen zugreifen.

Doch was ist der Unterschied zwischen Daten, Informationen und Wissen? Die folgende Grafik [8] versucht eine Antwort auf diese Frage zu geben.

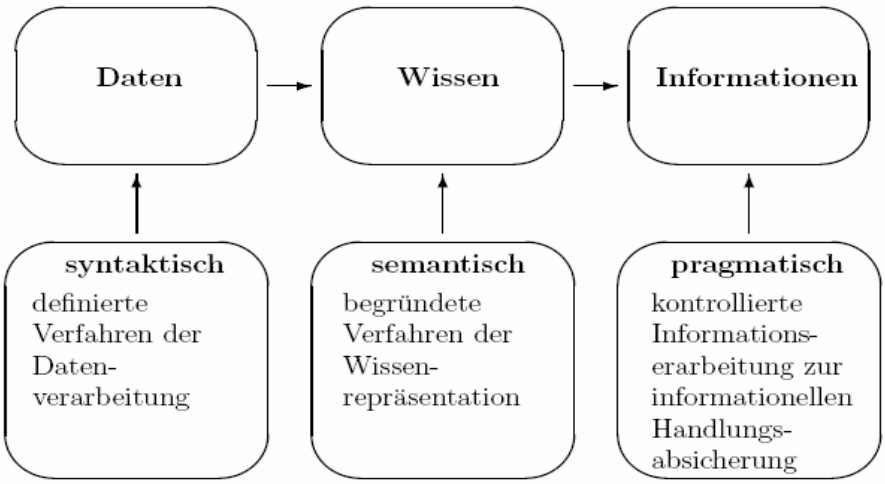

Abbildung 2: Unterschiede zwischen Daten, Wissen und Informationen.

Daten haben eine syntaktische Verankerung, was wiederum aus einer Datenbasis ein Konglomerat von Werten ohne jegliche Semantik macht. Sobald das Attribut der Semantik hin-

zukommt, spricht man von Wissen. Daraus folgernd können Datenbanksysteme nicht nur Daten, sondern auch Wissen enthalten, da über eine geeignete Modellierung der Rohdaten auch eine Semantik des jeweiligen Anwendungsgebiets im System entsteht. Analog dazu beinhaltet jedes IR-System Wissen.

Information ist auf der pragmatischen Ebene angesiedelt und kann als Teilmenge von Wissen angesehen werden, die von einem Benutzer in einer konkreten Situation zur Lösung eines Problems benötigt wird. Oft sind diese Informationen nicht verfügbar und müssen deswegen aus externen Quellen bezogen werden. Hierbei ist es die Aufgabe des Informationssystems die benötigten Informationen aus dem gesammelten Wissen zu extrahieren. Diese Transformation hebt allgemeines Wissen auf die höhere Stufe der Information und erzeugt dadurch einen Mehrwert. Informationen sind nicht nur spezialisierter, sondern auch flüchtiger als Wissen, das permanent zur Verfügung steht. In [8] wird die Beziehung zwischen Wissen und Information schlagwortartig und treffend durch die Formulierung „Information ist Wissen in Aktion“ ausgedrückt.

Abschließend soll hier auf ein einfaches Grundmodell für IR nach [9] verwiesen werden.

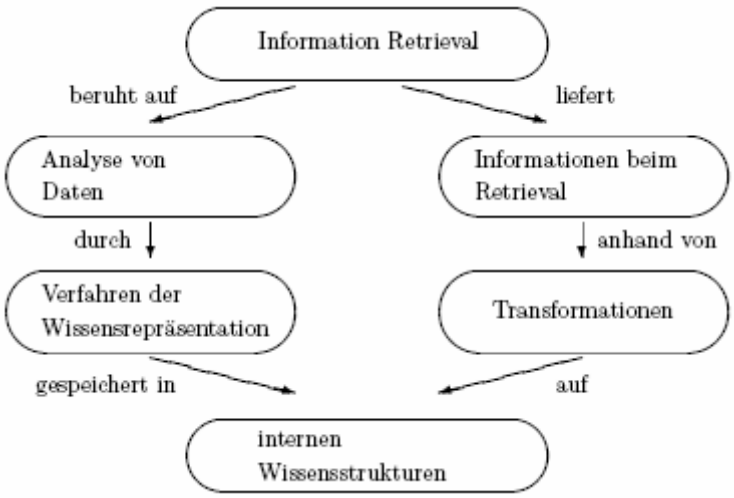

Abbildung 3: Ein Grundmodell für Information Retrieval

Die rechte Seite der Abbildung stellt den Retrievalprozess dar, während die linke Seite den Vorgang der Eingabe veranschaulicht. Beim Retrieval wird die benötigte Information durch Transformation auf dem gespeicherten Wissen erzeugt. Bei der Eingabe werden Daten analysiert und dann in gespeichertes Wissen überführt.

4. Beispiele angewandter Informationsvisualisierung

4.1. Cone Tree

Xerox Parc entwickelte 1991 im Rahmen des sog. „Information Visualizer“ Cone und Cam Trees zur Visualisierung von Hierarchien. Man unterscheidet die beiden Varianten durch die Anordnung der Hierarchieebenen. Beim Cam Tree sind die Ebenen wagerecht, beim Cone Tree senkrecht angeordnet. Wie so oft bei Visualisierungen von Hierarchien wird die Baumdarstellung herangezogen, wobei ein zusätzlicher Nutzen durch eine weitere räumliche Achse erzeugt wird.

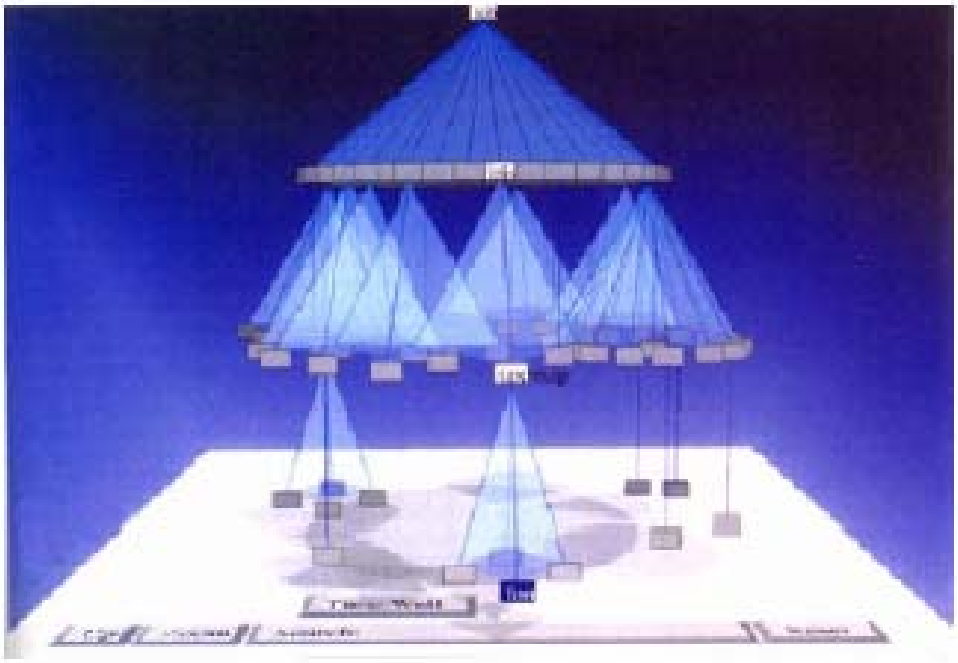

Abbildung 4: Visualisierung einer Hierarchie durch den Cone Tree.

Durch das Ausnutzen der räumlichen Tiefe lassen sich Kegel im Vordergrund oder im Hintergrund platzieren und bieten dadurch die Möglichkeit auf Informationen zu fokussieren. Damit gleichzeitig alle Strukturen sichtbar bleiben, sind die Kegel transparent. Ein Kegel bietet als dreidimensionales Visualisierungselement die Fähigkeit mehrere Kindsknoten abzudecken und ist damit prädestiniert für hierarchische Informationsstrukturen. Die Kindsknoten werden entlang des Umfangs der Kegelbasis angeordnet. Durch Vergrößern der Kegelbasis lässt sich die Anzahl der Kindsknoten erweitern, ohne dass übergeordnete Elemente in ihrer Struktur betroffen wären.

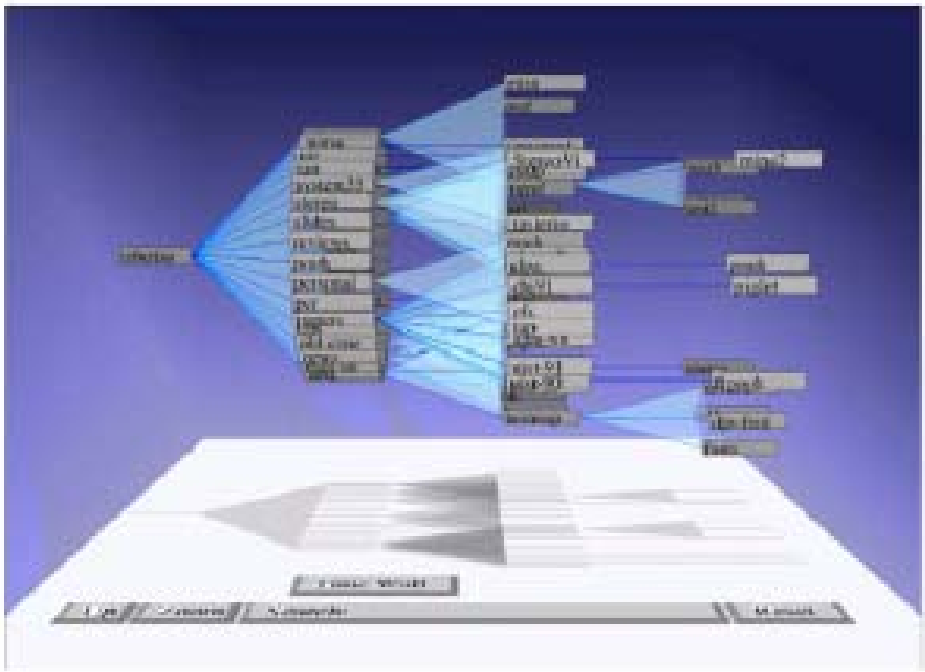

Abbildung 5: Cam Tree

Die einzelnen Kegel lassen sich interaktiv drehen. Verdeckte Elemente werden auf diese Weise in den Vordergrund geholt.

4.2. Collapsible Cylindrical Tree

Abbildung 6: Collapsible Cylindrical Tree

In [10] wird der *Collapsible Cylindrical Tree* vorgestellt, eine Anwendung zur Visualisierung und Navigation von Hierarchien mittlerer Größe. Die Oberfläche eines Zylinders repräsentiert eine Ebene der Hierarchie. Jeder Zylinder ist drehbar und ermöglicht dadurch die Selektion im Hintergrund liegender Elemente. Wenn der Benutzer ein Element auswählt und das Element kein Blatt in der Hierarchie darstellt, dann wird auf einem weiteren Zylinder die tiefer liegende Hierarchie abgebildet. *Collapsible Cylindrical Trees* bedienen sich außerdem der Teleskop-Metapher, indem die Zylinder der unteren Hierarchien im Umfang kleiner werden.

Collapsible Cylindrical Trees sind im Fall einer balancierten Baumstruktur theoretisch für $4{,}48 \cdot 10^8$ Knoten ausgelegt. Der Einsatz in der Praxis ist jedoch begrenzt auf mehrere Hundert bis mehrere Tausend Knoten.

4.3. Information Cube

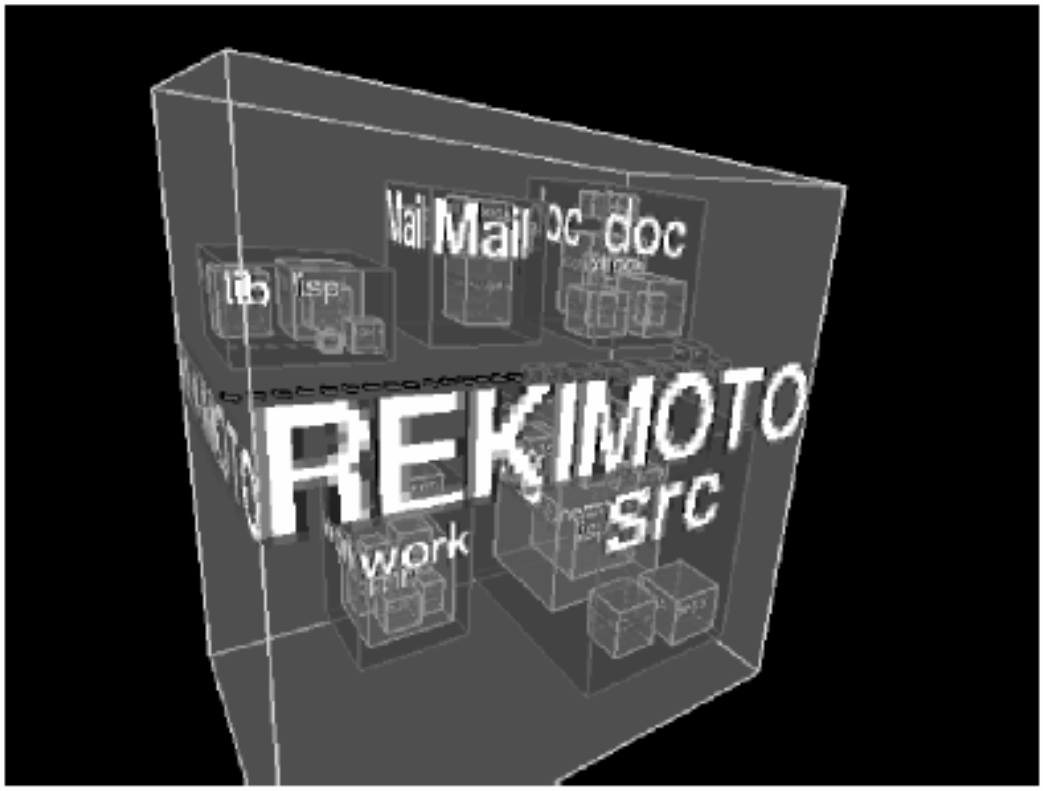

Abbildung 7: Information Cube

Der *Information Cube* nach [11] ist eine weitere Möglichkeit große hierarchische Informationsstrukturen zu visualisieren. Hierbei werden die Ebenen der Hierarchie durch ineinander geschachtelte Boxen abgebildet. Die Ebenentiefe wird zum einen durch das Ineinanderschachteln der Quader erzeugt, zum anderen durch den Einsatz von unterschiedlichen Transparenzgraden. Die oberste Hierarchieebene, sprich die alles umgebende Box, ist halbtransparent und bewirkt, dass tiefer liegende Ebenen noch sichtbar bleiben. Die „Cubes" sind durch eine Aufschrift gekennzeichnet und ermöglichen dadurch eine Auswahl des gesuchten Informationselements bzw. der gewünschten Hierarchieebene. Die Würfelmetapher erlaubt zudem den Einsatz weiterer beliebiger geometrischer Objekte in den Würfeln. Die Boxen lassen sich rotieren und tief liegende Elemente lassen sich über einen virtuellen Strahl auswählen. Der *Information Cube* ist für große hierarchische Informationsstrukturen ausgelegt. Im vorliegenden Beispiel (siehe Abbildung 7) werden 50 Verzeichnisse mit 1500 Dateien visualisiert und verwaltet.

4.4. Information Corridor

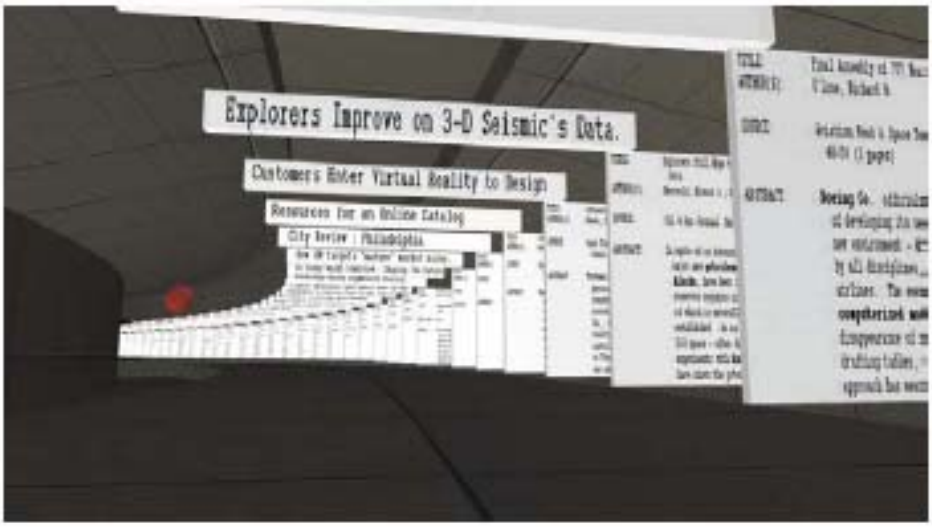

Abbildung 8: Information Corridor

Lineare und eindimensionale Informationsstrukturen lassen sich mit dem *Information Corridor* nach [12] darstellen. Hierbei bietet sich besonders das Durchsuchen einer Datenbank an. Die Darstellung der Datensätze erfolgt durch Karteikarten, die in eine Tunnelarchitektur eingegliedert sind. An der Decke des Tunnels befinden sich Schilder mit der Aufschrift des Titels einer Karteikarte. Somit kann sich der Nutzer durch den „Corridor" bewegen und anhand der Schilder eine Auswahl treffen, indem er die zugehörige Karteikarte per Drag & Drop in die Mitte zieht. Je größer die Datenbank ist, desto länger muss auch der Tunnel werden. Die Anwendung stößt hierbei schnell an die Grenzen der effizienten Benutzbarkeit, da kein User eine „Tunnelfahrt" von mehreren Tausend Karteikarten erduldet. Für kleine Datenbestände ist der *Information Corridor* eine attraktive Visualisierungstechnik.

4.5. Perspective Wall

Zur sequentiellen Darstellung von Dateien über eine Zeitachse lässt sich die *Perspective Wall* benutzen. Hierbei werden Dateien nach ihrem Erstellungsdatum angeordnet und auf einer Wandseite visualisiert. Die Wand kann interaktiv verschoben werden, so dass jeweils andere Zeitabschnitte in den Vordergrund gelangen. Mit Hilfe der perspektivisch verzerrten Wände können unwichtige Informationen aus dem Fokusbereich genommen werden. Das bietet Vorteile gegenüber der klassischen und flachen Darstellung, da die Darstellungsfläche effektiver genutzt werden kann. Die Wand selber ist vertikal in mehreren Flächen unterteilt, wobei die Flächen untereinander korrelieren können [13]. Somit können relevante Daten kohärent und synchron dargestellt werden. Jedoch ist die Wand in der Höhe beschränkt und bietet nur in der Länge theoretisch unbegrenzte Darstellungsfläche.

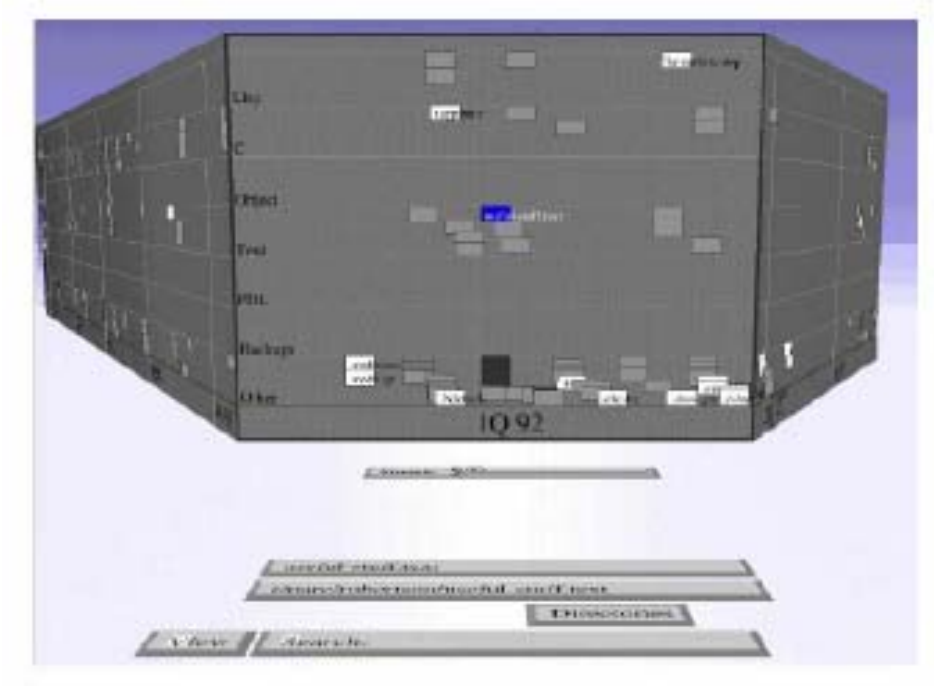

Abbildung 9: Perspective Wall

Zusammenfassend lässt sich sagen, dass die *Perspective Wall* gut geeignet ist für lineare Daten mit zeitlichem Bezug.

4.6. Documentfinder

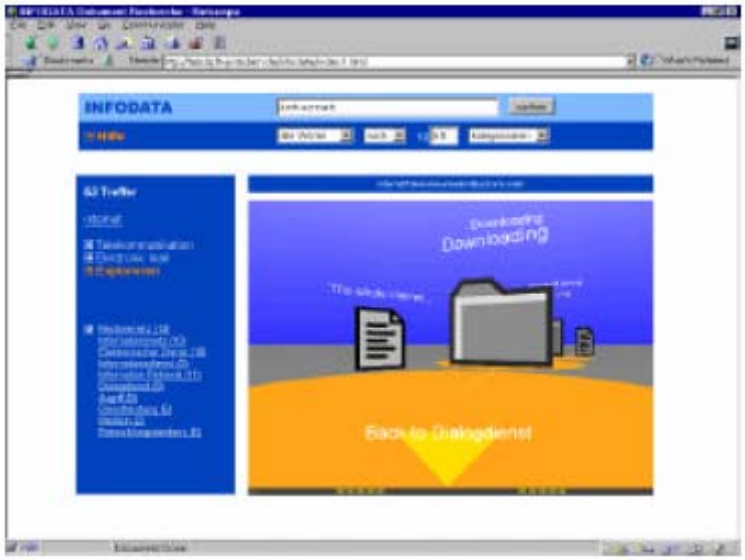

Abbildung 10: Document-Finder

Der *Document-Finder* [2] unterstützt den Benutzer bei der Informationssuche im Web. Die HTML-Ergebnislisten mit einer großen Anzahl von Einträgen, die aufgrund einer Stichwortsuche in Suchmaschinen generiert werden, sind oft problematisch in ihrer Strukturierung und inhaltlichen Aufbereitung. Der *Document-Finder* liefert auf eine Suchanfrage eine strukturierte Treffermenge, die in einer dynamisch generierten Informationslandschaft dargestellt wird. Im vorliegenden Beispiel (siehe Abbildung 10) bildet der *Document-Finder* eine visuelle Schnittstelle für die bibliographische Datenbank *Infodata*, die ca. 80.000 Einträge verwaltet. Die Abbildung zeigt den virtuellen Informationsraum, der durchflogen werden

kann und somit eine Exploration der Dokumenttreffer erlaubt. Es lässt sich jedoch bereits an der Benutzeroberfläche des *Document-Finder* erkennen, dass man auch hier nicht ohne klassische Benutzereingabeelemente auskommt, wie z.B. eine Suchmaske oder eine Textbox zur Eingabe des Retrievalpfades.

4.7. Newsmap – Newsbrowser

Abbildung 11: Newsmap-Nachrichtenbrowser

Der *Newsmap-Nachrichtenbrowser* der Firma Cartia[14] erfasst und kategorisiert Nachrichten und mappt diese als Themenbereiche in eine Dokumentkarte. Die Karte erlaubt es, wie eine topographische Karte aus der Geodäsie, Höhenunterschiede durch unterschiedliche Farben darzustellen. Erhöhungen oder „Berge“ repräsentieren hohe Datenkonzentrationen. Höhenlinien bilden ein Konglomerat von zusammengehörigen Domänen.

4.8. MineSet

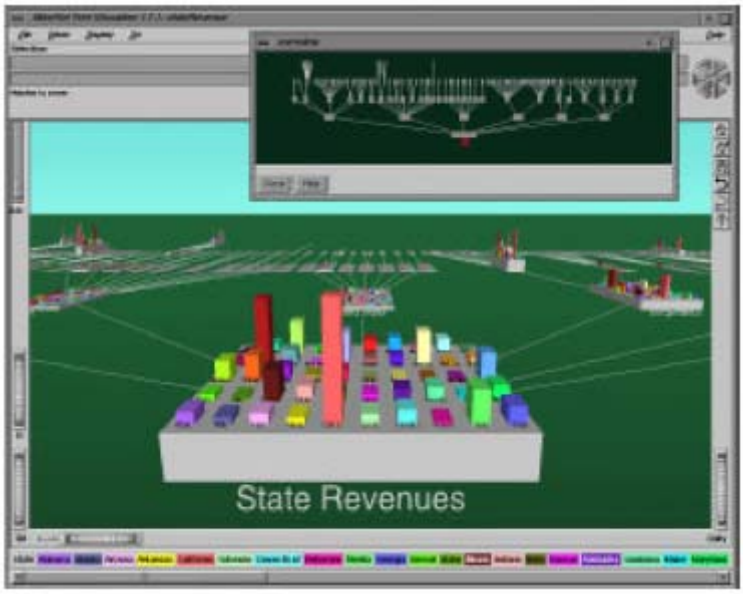

Abbildung 12: MineSet von SGI

Im Data-Mining-System *MineSet* von SGI[15] werden hierarchische Datenstrukturen in Form einer 3D-Informationslandschaft dargestellt. Der Benutzer kann durch den Datenraum fliegen, wobei ein zweidimensionales Übersichtsfenster die Orientierung erleichtert. Das vorliegende Beispiel (siehe Abbildung 12) visualisiert die Einnahmen der einzelnen Staaten der USA in diversifizierten Kategorien. Die unterschiedlichen Farben repräsentieren die Staaten, während die Höhe der Balken proportional zur Höhe der Einnahmen ist.

4.9. Polyarchie Visualisierung

Robertson, Cameron et al [16] entwickelten diese Anwendung um so genannte Polyarchien zu visualisieren. Der Begriff der Polyarchie wird dort als eine Menge von Hierarchien beschrieben, die wiederum mindestens einen Knoten gemeinsam verwenden. Die traditionelle Darstellung von Hierarchien ist nach ihren Ausführungen ungenügend, um die Beziehungen zwischen den Elementen verschiedener Hierarchien hinreichend anschaulich zu visualisieren. Selbst moderne Visualisierungsformen werden dieser Anforderung nicht gerecht.

Die Autoren beschreiben eine Reihe von möglichen Darstellungsformen für dieses Problemfeld, welche alle auf dem Konzept des dort entwickelten „Visual Pivot“ beruhen. Hierfür wählt der Benutzer über einen Browser eine konkrete Hierarchie aus und bestimmt ein Element als sog. Pivot-Element. Dies ist nun das designierte Element, was hinsichtlich Verbindungen zu anderen Hierarchien untersucht werden soll. Möchte sich der Benutzer nun eine verbundene Hierarchie anzeigen lassen, so wird eine animierte Transformation präsentiert, welche den Übergang von der alten in die neue Hierarchie visualisiert. Das vorher ausgewählte Pivot-Element bestimmt die Stelle, an der die beiden Hierarchien ineinander übergehen (siehe Abbildung 13). Um mehr als zwei Hierarchien gleichzeitig zu untersuchen, entwickelten die Autoren eine weitere Darstellungsform, welche in Abbildung 14 zu sehen ist. Dort wird das ausgewählte Pivot-Element durch Linien über die gewählten Hierarchien verbunden.

Die Abbildungen zeigen als Beispiel parallele Organisationshierarchien in einem Unternehmen. Als Hierarchien werden Geschäftseinheiten, Führungsstruktur und Büroanordnung angeführt. In Abbildung 13 wird die Hierarchie der Führungsstruktur über eine paral-

lele Verschiebung in die Hierarchie der Geschäftseinheiten überführt. Als Pivot-Element wird ein bestimmter Mitarbeiter des Unternehmens gewählt, um dessen Rolle in beiden Hierarchien zu bestimmen.

Die Autoren implementierten eine Reihe von möglichen Animationen und führten empirische Tests mit Benutzern durch um eine optimale Konfiguration für die praktische Anwendung zu finden. Als Ergebnis wurde die in Abbildung 13 dargestellte Transformation für als gut geeignet befunden.

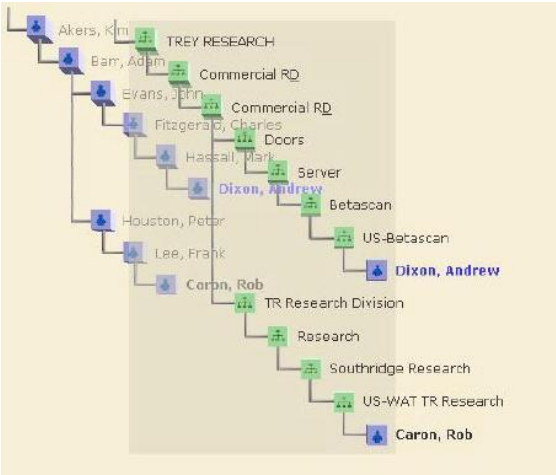

Abbildung 13 Transformation der Führungsstruktur zu Geschäftseinheiten

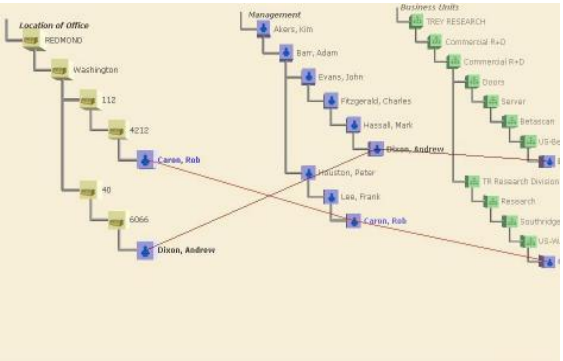

Abbildung 14 Multiple Hierarchien

4.10. Visualisierung von Geschäftsprozessen

Bei der Optimierung von Geschäftsprozessen in einem Unternehmen werden seit langem sog. Prozessgraphen zur Visualisierung verwendet. Diese werden in der Regel im 2D-Raum dargestellt und bestehen aus einer Reihe von Knoten, welche Arbeitsstationen beschreiben, in denen eine Menge bestimmter Aufgaben erfüllt werden können. Kanten verbinden diese Arbeitsstationen entsprechend der in diesem Modell möglichen Arbeitsabläufe. Eine Menge von Regeln gibt an, welche Arbeitsschritte zur Lösung eines Aufgabentyps notwendig sind. Eine konkrete Aufgabe durchläuft dann alle Arbeitsstationen, welche für ihre Bearbeitung notwendig werden, in der durch die Graphenstruktur vorgegebenen Weise. Anhand von Simulationsverfahren lassen sich solche Modelle auf Auslastung der einzelnen Stationen und dabei entstehende Engpässe untersuchen. Mit den Ergebnissen kann man das Modell dann nach den gegebenen Anforderungen optimieren (minimale mittlere Durchlaufzeit, minimale Anzahl von Stationen, ...).

Die Autoren Schönhage, van Ballegooij und Eliens haben in [17] ein System entwickelt und untersucht, welches die Ergebnisse eines solchen Simulationsverfahrens in einem 3D-Modell darstellt. In Abbildung 15 ist das Ergebnis dieser Visualisierung dargestellt. Die großen, grauen Kugeln repräsentieren Arbeitsstationen, die kleineren farbigen Kugeln stellen die zu bearbeitenden Aufgaben dar. Die Quader stellen Warteschlangen dar, welche als Zwischenablage für Aufgaben dienen. Wie in der Abbildung zu sehen ist, sind Aufgaben immer einer Arbeitsstation oder einer Warteschlange zugeordnet. Die Abbildung stellt insgesamt das Zwischenergebnis eines Simulationslaufs dar. Um die Ergebnisse zu analysieren, kann z.B. ein Histogramm für die verschiedenen Warteschlangen angezeigt werden (siehe Abbildung 16). So kann sich der Benutzer einen Überblick über Auslastung und

Wartezeiten verschaffen. Ein graphischer Drill-Down, also die Veränderung von Aggregations-Kriterien für die Darstellung, ist möglich.

Die Autoren stellen gegenüber früheren 2D-Anwendungen als Vorteil heraus, mehr Informationen gleichzeitig präsentieren zu können und dem Benutzer reichhaltigere Navigations- bzw. Manipulationsmöglichkeiten zu gestatten. Als Nachteile wurden jedoch eine mäßige Akzeptanz der Benutzer und eine komplexere Bedienung erwähnt.

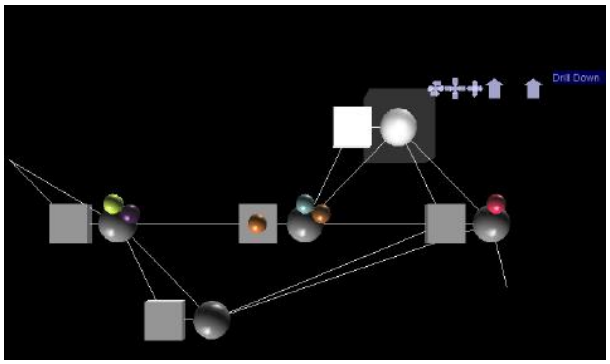

Abbildung 15 Prozessgraph in 3D

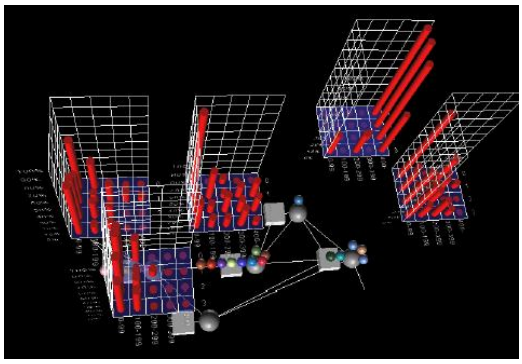

Abbildung 16 Prozessgraph mit Histogramm

4.11. Visualisierung der Entwicklung von WWW-Strukturen

In [18] beschreiben Chi et. al. ein System zur Visualisierung der dynamischen Entwicklung von Web-Inhalten. Es soll hier die Beziehung zwischen Inhalten, Topologie und Benutzerverhalten über der Zeit in einer innovativen Visualisierung genannt *Time Tube* dargestellt werden. Ein Benutzer soll anhand der Visualisierung feststellen können wie eine Web-Site strukturiert ist, wie andere Benutzer darauf zugreifen und wo sich interessante Änderungen befinden könnten. Einem Administrator soll gezeigt werden, wie sich Redesigns auf das Benutzerverhalten auswirken und für welche Seiten sich Besucher der Site wirklich interessieren.

Zur Visualisierung der Topologie werden sog. Disk-Trees verwendet. Hierfür wird zunächst ein aufspannender Baum der Graphen-Repräsentation der Web-Site, ausgehend von der Startseite berechnet. Dieser wird dann auf eine Konstruktion, bestehend aus einer Menge von konzentrischen Kreisen, abgebildet, indem jede Ebene des Baums einem Kreis, aufsteigend von innen nach außen, zugeordnet wird. Die Kanten werden in die Räume zwischen den Kreisen gezeichnet. In Abbildung 17 sieht man vier solcher Disk-Trees angeordnet in der erwähnten *Time Tube*. Diese repräsentiert die Entwicklung der Site (in diesem Fall die Web-Site von Xerox-Parc) über eine Spanne von vier Wochen. Geänderte Elemente oder Links werden in zeitlich nachfolgenden Disk-Tree farblich markiert. Obwohl die Disk-Tree Struktur inhärent zweidimensional ist, wird für die Präsentation der *Time Tube* der 3D-Raum verwendet. Die Autoren begründen dies mit der Möglichkeit verschiedene *Time Tubes* flexibel im Raum anzuordnen. Eine Möglichkeit wäre z.B. eine Fluganimation durch hintereinander angeordnete *Time Tubes* zu realisieren. Die Autoren haben eine Reihe von Interaktionsmöglichkeiten implementiert, welche dem Benutzer eine Auswertung der umfangreichen Darstellung erleichtern sollen.

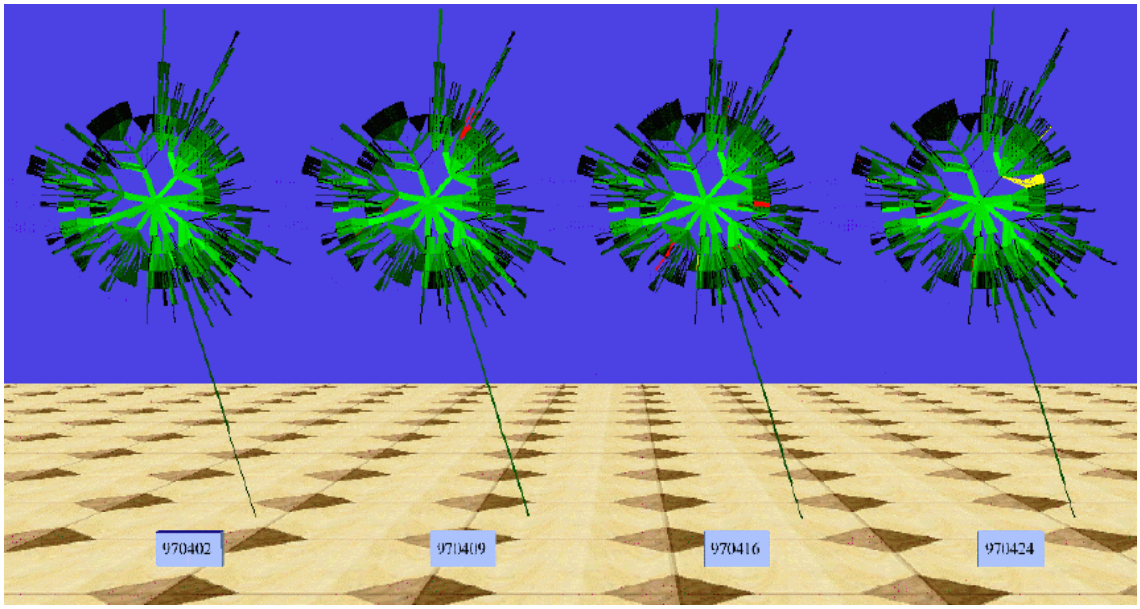

Abbildung 17 Time-Tube einer Web-Site über 4 Wochen

Er kann sich mit einem Mausklick die normalerweise im Raum angeordneten Disk-Trees nebeneinander anzeigen lassen (siehe Abbildung 17). Er hat auch die Möglichkeit eines graphischen Drill-Downs, also die detailreichere Anzeige eines Bereichs des Baumes. Wenn eine interessante Seite vermutet wird, kann der Benutzer das Programm veranlassen direkt einen Web-Browser mit der gewählten Seite zu starten.

5. Beispielanwendung: *INVENTORY 3D*

5.1. Ziele und Anforderungen

Mit der Beispielanwendung *INVENTORY 3D* soll exemplarisch gezeigt werden, wie Visualisierungsapplikationen im Bereich der Lagerverwaltung zum Einsatz kommen können. Das primäre Ziel für *INVENTORY 3D* war das Finden einer geeigneten Metapher zur Darstellung der für die Inventarverwaltung wichtigen Größen, wie z.B. Artikelbezeichnung, Artikelmenge, Preis und Reorderlevel, wobei dieser angibt, ab welcher Artikelanzahl neue Artikel geordert werden sollten.

Das sekundäre Ziel war die Erstellung einer benutzerfreundlichen Schnittstelle zwischen Benutzer und Programm. Hierbei sollten vor allem die eingabevereinfachenden Funktionen einer dreidimensionalen Oberfläche ausgenutzt werden, da die visuelle Aufnahme von Informationen beim Menschen am günstigsten und effektivsten ist. Um die Eingabemöglichkeiten des Users zu erweitern, wurde außerdem das Man-Machine-Interface mit der Fähigkeit der Spracheingabe ausgestattet. Es soll möglich sein wichtige Befehle über Spracheingabe auslösen zu können, damit nicht immer eine Tastatur notwendig ist.

INVENTORY 3D ist eine Anwendung, die sich gut über eine Client/Server-Architektur realisieren lässt. Sie umfasst also nicht nur die Desktopanwendung, die für die Visualisierung verantwortlich ist, sondern auch eine dazugehörige Datenbank mit allen Artikeln und eine Webapplikation für die mobilen Clients (Laptop, PDA, Smartphone und im ungünstigsten Fall auch Handy).

Abbildung 18: Architektur von *INVENTORY 3D*

Man könnte sich folgendes Szenario vorstellen: Die mit PDA's ausgestatteten Lagerarbeiter nehmen die Artikelbestände auf und speichern diese dann in einer Datenbank. Die Daten werden hierbei über das WLAN[3] übertragen und durch das SOAP-Protokoll kodiert. SOAP[4] bietet sich an, da es für sog. Webservices vorgesehen ist, die zunehmend bei Client/Server-Architekturen zum Einsatz kommen. Somit ist die Artikeldatenbank ständig bereit auf Anfragen der Lagerverwaltung mit aktualisierten Daten zu antworten. Die Lagerverwaltung, hierbei kann es sich auch um mehrere zuständige Personen handeln, hat die Möglichkeit sich diese Daten in visualisierter Form anzuschauen und zu analysieren. Um die visualisierten Daten allen Mitarbeitern zugängig zu machen, könnte die Desktopoberfläche von *INVENTORY 3D* über Beamer an die Wände der Lagerhalle projiziert werden. Zur Bedienung würde dann ein Headset ausreichen, da alle wichtigen Befehle zur Analyse über Spracheingabe erfolgen.

Obwohl *INVENTORY 3D* nur zu Demonstrationszwecken vorgesehen ist, sollte die Anwendung grundsätzlich erweiterbar und wartbar sein. Aus diesem Grund werden bei der Konzeption die Komponentenorientierung und die dazugehörigen Unittests (siehe Kapitel 5.3.2.1.4 Testing Framework NUnit) als wichtige Anforderungen genannt. Die Szenengenerierung und auch das Interface zum Cortona VRML Viewer sind als Library implementiert und lassen sich für weitere Anwendungen modifizieren, integrieren, und auch refaktorieren, da die Konsistenz durch die Unittests ausreichend gegeben ist.

[3] Wireless LAN
[4] Simple Object Acces Protocol

5.2. Architektur

Die Architektur von *INVENTORY 3D* mit ihren einzelnen Komponenten soll nun im Einzelnen erklärt werden. Der Zusammenhang der Module wird anhand der folgenden Abbildung deutlich.

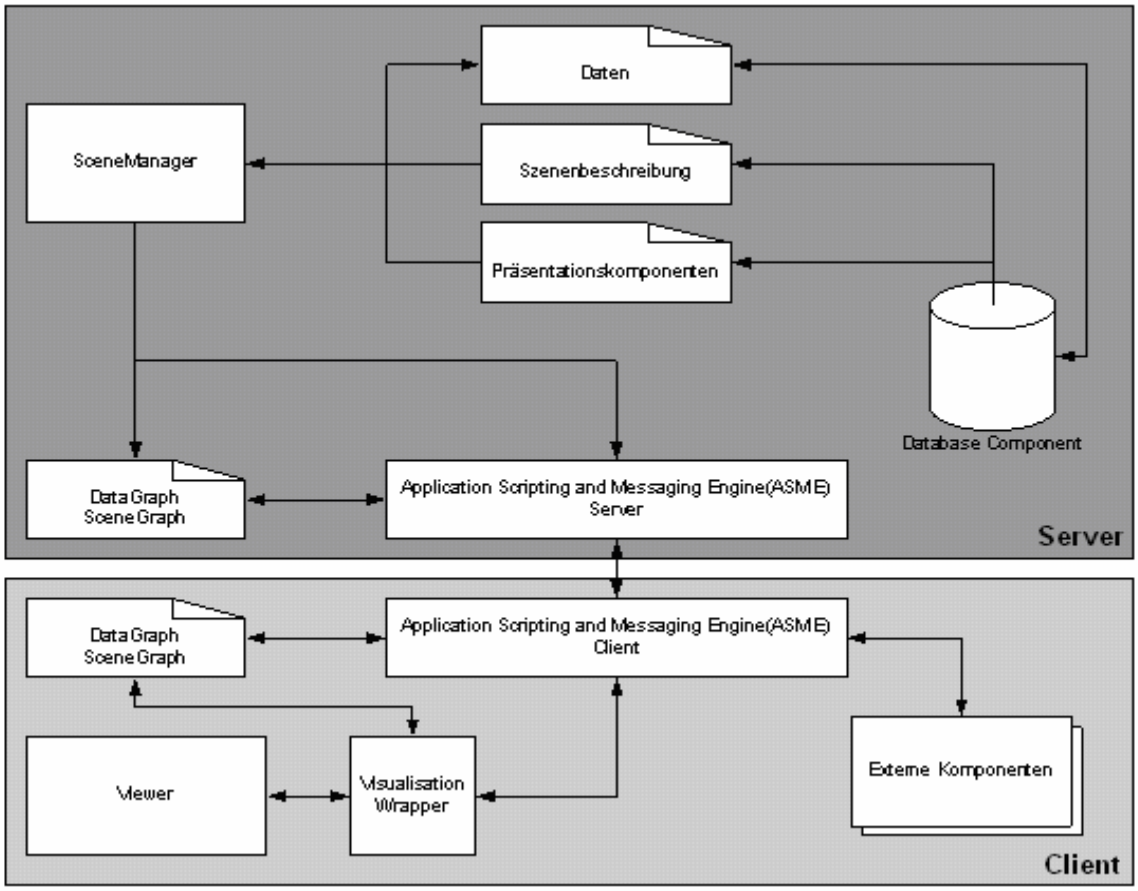

Abbildung 19: Diagramm der Client/Server-Architektur

5.2.1. Präsentationskomponenten

Die Präsentationskomponenten sind die Grundbausteine für jede Visualisierung und stellen jeweils einen Teil einer Szene dar. Sie enthalten sowohl eine Beschreibung ihrer visuellen Struktur als auch eine Verhaltensbeschreibung, in der gewisse komponentenspezifische Funktionalität hinterlegt wird.

5.2.2. Szenenbeschreibung

In einer Szenenbeschreibung finden sich die Regeln zum Aufbau einer informationsvisualisierenden 3D-Szene. Dort sind alle Angaben über die zu extrahierenden Daten, die zu verwendenden Präsentationskomponenten und über die Abbildung der Daten auf die Präsentation.

5.2.3. Database Component

Diese Komponente von *INVENTORY 3D* ist zuständig für den Zugriff auf alle Daten die in der Anwendung verwendet werden. Einerseits dient sie als „Object-Repository“, d.h. alle

Szenenbeschreibungen und Präsentationskomponenten werden hier hinterlegt. Andererseits wird hierüber der Zugriff auf die zu visualisierenden Daten (sprich Artikel) abgewickelt.

5.2.4. SceneManager

Diese Komponente ist für den Aufbau einer Szene verantwortlich. Ihr wird eine Szenenbeschreibung übergeben, welche ausgeführt bzw. interpretiert wird. Die SceneManager Komponente sorgt für die Instanziierung der Präsentationskomponenten und die Extraktion der Daten. Als Ausgabe wird der so genannte SceneGraph und DataGraph generiert. Beim Wechsel einer Szene ist der SceneManager auch für den korrekten Abbau der bestehenden Szene und ein eventuelles Zurückschreiben der Daten verantwortlich.

5.2.5. SceneGraph

Der SceneGraph repräsentiert eine generische Beschreibung einer 3D-Szene und wird von allen Komponenten des Frameworks als Schnittstelle zur Manipulation der visuellen Darstellung genutzt.

5.2.6. DataGraph

Der DataGraph enthält alle Informationen über die Herkunft der dargestellten Informationen. Es ist hinterlegt, wie und wohin geänderte Information zurück geschrieben werden können. Außerdem werden die Abhängigkeiten zwischen den einzelnen Daten und ihre Zuordnung zum SceneGraph dargestellt.

5.2.7. Application Scripting and Messaging Engine (*INVENTORY 3D* - ASME)

Diese Komponente ist für die Verwaltung und die Ausführung von Verhaltensskripten zuständig, welche von den Präsentationskomponenten und aus der Szenenbeschreibung stammen. Die Ausführung solcher Skripte wird durch ein Nachrichten- oder Messagesystem ausgelöst. Messages werden durch Benutzerinteraktion oder Skriptausführung generiert.

5.2.8. Visualization-Wrapper & Viewer

Beim Viewer handelt es sich um eine konkrete Instanz einer 3D-Viewer Technologie. Der Viszualization-Wrapper hat die Aufgabe diesen Viewer mit einer generischen Schnittstelle zu versehen. Der Viewer-Wrapper konvertiert das generische Szenenformat von *INVENTORY 3D* in das spezifische Format des Viewers. Er stellt außerdem ein einheitliches Konzept zur Verarbeitung von Benutzeraktionen zur Verfügung.

5.2.9. Navigation

Bei den Navigationskomponenten handelt es sich um entwickelte graphische Komponenten, die dem Benutzer die Bedienung der gesamten Anwendung ermöglichen. Es kann sich dabei beispielsweise um eine Menüleiste oder eine Toolbar handeln.

5.2.10. Konfiguration der Komponenten

In der oben dargestellten Client-Server Konfiguration wurden die wesentlichen Komponenten zu einem Visualisierungsserver zusammengefasst. Dieser hat die Aufgabe des Szenenaufbaus und des Datenzugriffs. Der Client ist für die Darstellung und für die Interaktion mit der Szene verantwortlich. Der Vorteil dieser Aufteilung liegt darin, dass nur der Server für den Datenzugriff konfiguriert sein muss. Auf Szenebeschreibungen und Präsentationskomponenten kann schnell zugegriffen werden, da diese auf dem Server gelagert

sind. Außerdem wird der unter Umständen aufwendige Prozess der Szenengenerierung vom Server ausgeführt, der hierfür besondere Systemressourcen bereithalten kann.

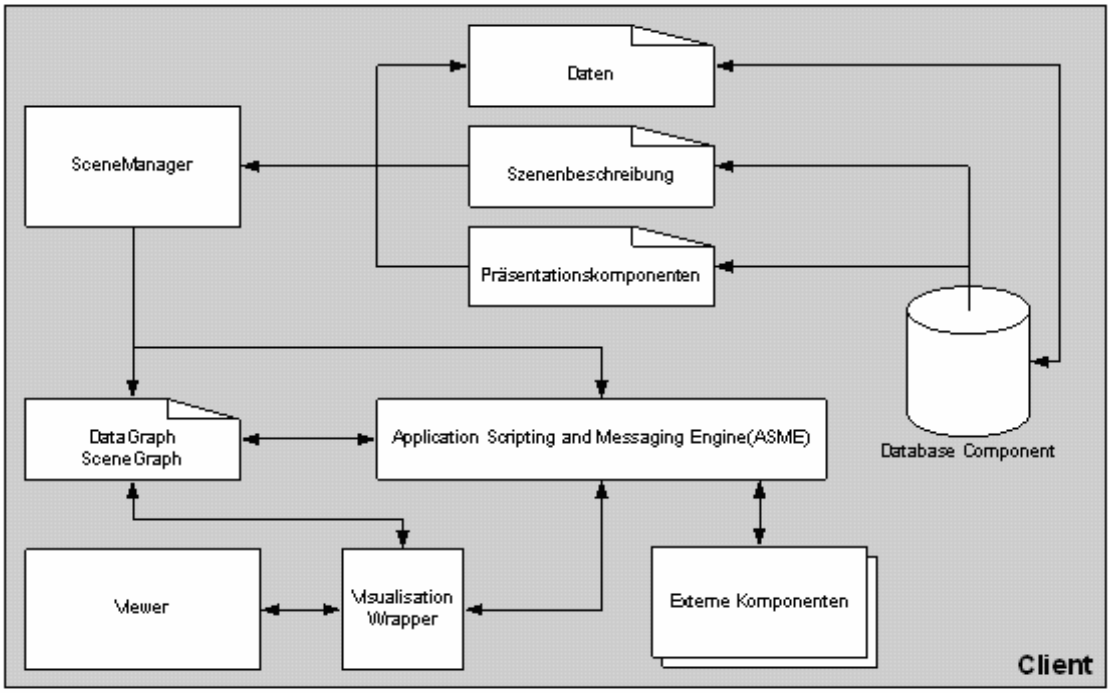

Abbildung 20: Visualisierungskomponenten des Desktop Clients

Anhand der Abbildung 20 lässt sich bereits erkennen, welche Komponenten wichtige Funktionen innerhalb von *INVENTORY 3D* ausführen. Besonders der SceneManager und die Application Scripting and Messaging Engine (ASME) sind Schlüsselkomponenten der Anwendung. Die ASME dient jedoch lediglich zum Transfer der Daten zwischen Server und Client (et vice versa) und ist durch einen Webservice realisiert. Die Aufgaben des SceneManager sind weit reichender und umfassen fast die gesamte Funktionalität von *INVENTORY 3D*.

5.2.11. Datenstrukturen

5.2.11.1. SceneGraph

Der SceneGraph hat die Aufgabe eine 3D-Szene in einer von einer speziellen 3D-Technologie unabhängigen Weise zu repräsentieren. Das prinzipielle Vorgehen eine 3D-Szene durch einen so genannten Szenengraph zu repräsentieren ist nicht neu und wird in vielen Produkten erfolgreich eingesetzt. Ein bekanntes Beispiel ist der VRML97-Standard [19]. Die Idee hinter einem solchen Szenegraph-Modell besteht darin, alle Objekte einer 3D-Szene in einer Baumstruktur anzuordnen. Objekte einer 3D-Szene sind z.B. einzelne Formen (Kugeln, Zylinder, Polygone, …) oder auch Lichtquellen und Betrachtungspunkte. Jeder dieser Knoten besitzt eine Reihe von Attributen, die seine Darstellung definieren. Eine Kugel besitzt z.B. das Attribut Radius. Die Anordnung der Objekte im Baum entspricht nun der Positionierung dieser Objekte relativ zu lokalen Koordinatensystemen. Ein lokales Koordinatensystem entsteht jeweils durch die Transformation des übergeordneten Koordinatensystems. Dabei wird eine neue orthogonale Basis für das lokale Koordinatensystem festgelegt, die durch eine Drehung, eine Verschiebung und eine Skalierung des vorangegangenen Systems definiert wird. Die Positionen der visuellen Objekte in einem Szenengraph sind immer fest vorgegeben und beziehen sich immer auf das letzte übergeordnete lokale Koordinatensystem. Bei einer Kugel liegt der Ursprung z.B. immer in der Mitte

des aktuellen Koordinatensystems. Die Positionierung der Objekte wird lediglich durch die Transformation des zugeordneten Koordinatensystems erreicht. In einem Szenengraph wird eine Koordinatentransformation durch einen sog. Transformationsknoten eingeleitet. Ein Transformationsknoten kann dabei beliebig viele Nachfolger besitzen.

So entsteht eine Baumstruktur mit Transformationsknoten als innere Knoten und allen anderen Objekten als Blätter. Der Vorteil einer solchen Darstellung besteht darin, dass bei der dynamischen Generierung einer 3D-Szene keine absoluten Positionen bekannt sein müssen. Einzelne Routinen können in einem lokalen Koordinatensystem arbeiten, ohne dessen globale Position kennen zu müssen. Für die spätere Implementierung muss berücksichtigt werden, dass sich das gewählte Format leicht in bestehende proprietäre Formate konvertieren lässt und dass keine Knotentypen verwendet werden, die in vielen Szenegraph-Formaten nicht bekannt sind.

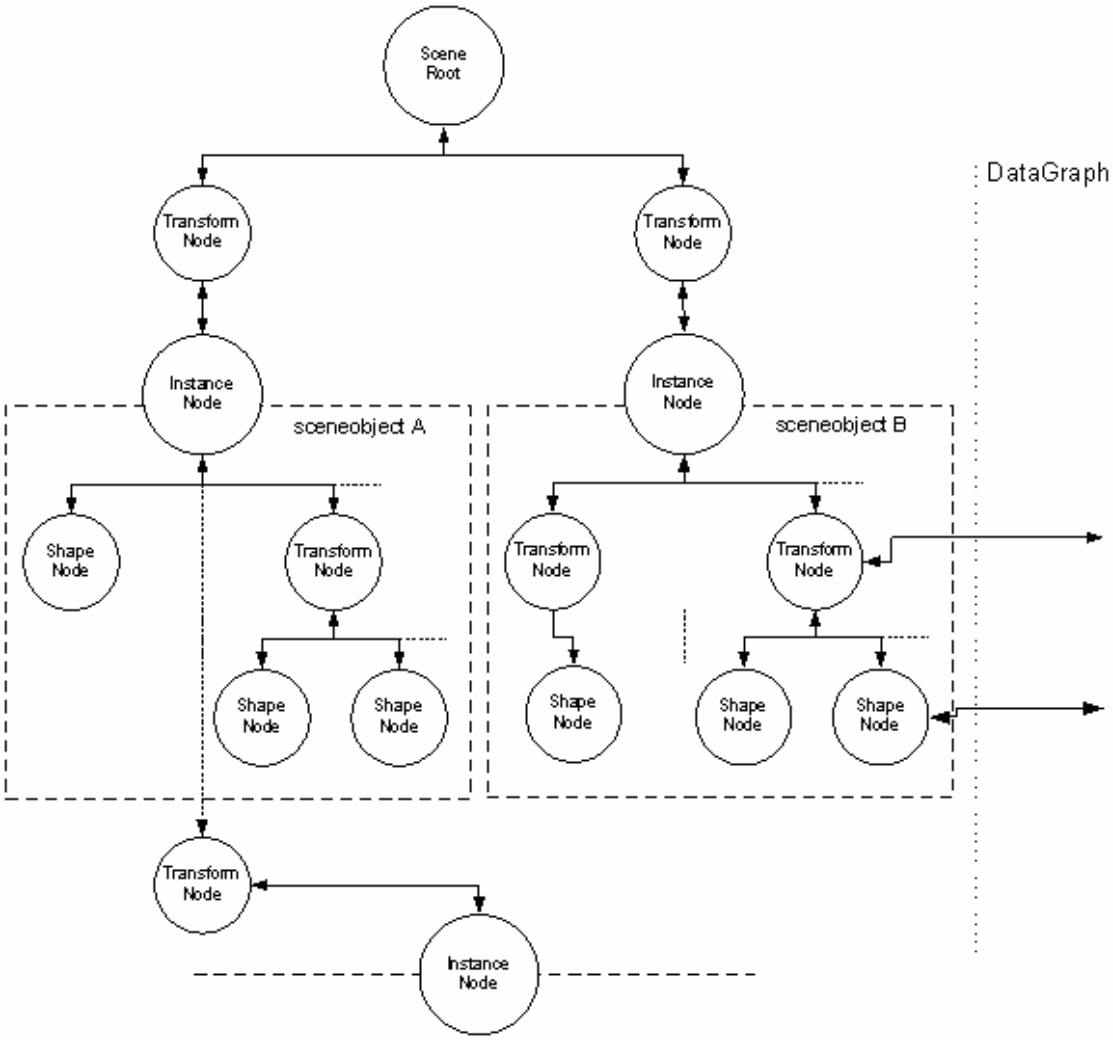

Abbildung 21: Schematische Darstellung des Szenegraphen

Da eine 3D-Szene in dieser Anwendung aus bestehenden Präsentationskomponenten zusammengesetzt wird, muss der Szenengraph um einen Knotentyp angereichert werden um diese Tatsache ausdrücken zu können. In Abbildung 21 ist schematisch ein Szenengraph dargestellt, wie er für die Anwendung verwendet werden soll. Es wird hier, im Gegensatz zu VRML97 eine explizite Wurzel des Baumes angegeben (Scene Root), was die Weiterverarbeitung erleichtert. Die Verwendung einer Präsentationskomponente wird durch einen Instance Node angezeigt. Er dient als Wurzel für die Teilszene, die durch die Komponente definiert wird. Außerdem kann in diesem Knoten eine Referenz auf die verwendete Klasse

angegeben werden. Der Instance Node führt selbst keine weitere Transformation des Koordinatensystems durch, er gruppiert lediglich die Knoten der Präsentationskomponente. Die verschiedenen Instanzen der Präsentationskomponenten werden durch während der Szenengenerierung erzeugte Transformationsknoten zusammengefügt. Auf einen Instance Node können noch weitere Instance Nodes folgen, indem ein Transformationsknoten dazwischen gehängt wird.

In der Abbildung ist bereits die Verbindung zum DataGraph angedeutet. Die einzelnen Attribute der Knoten des SceneGraph können so mit Datenwerten aus dem DataGraph verknüpft sein. Dies ermöglicht es später auf Änderungen der Daten zu reagieren und die 3D-Szene anzupassen.

5.2.11.2. DataGraph

Im DataGraph werden die zu visualisierenden Daten mit einer Reihe von Metainformationen hinterlegt. In Abbildung 22 ist die Struktur eines solchen DataGraphs mit typischen Knoten schematisch dargestellt.

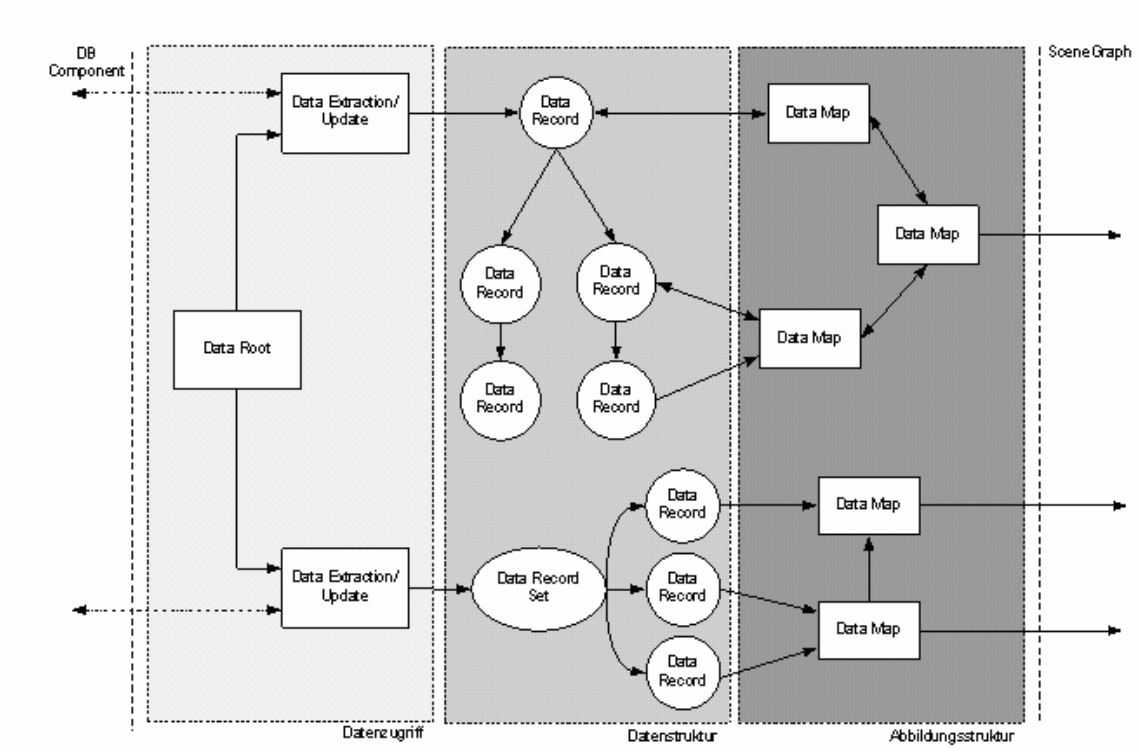

Abbildung 22: Schematische Darstellung des Datengraphs

Er kann in drei logische Teile partitioniert werden: Datenzugriff, Datenstruktur und Abbildungsstruktur.

5.2.11.2.1. Datenzugriff

Im Teil des Datenzugriffes wird definiert welche Daten in der Anwendung verwendet werden sollen. Dazu werden sog. Data Extraction/Update Knoten verwendet. In diesen Knoten wird angegeben woher die Daten extrahiert werden sollen. Je nach Ausgestaltung der Datenquelle kann diese Angabe unterschiedlich ausgestaltet sein. Im Fall einer relationalen Datenbank könnte es sich bei den Extraktionsdaten um einen SQL-Befehl zusammen mit einem Datenbank Login handeln. Handelt es sich hingegen um eine Datei in der Daten beispielsweise im XML-Format gespeichert sind, so wird es sich bei den Informationen um die URL des Dateinamens handeln und eventuell einen XPath-Ausdruck, der einen Teil-

baum des XML-Dokumentes spezifiziert. Da es sich bei der *INVENTORY 3D* nur um eine Demonstrationsanwendung handelt, wurde lediglich eine Schnittstelle für Datenbanken implementiert. Die Knoten enthalten außerdem Informationen darüber, an welche Stelle geänderte Daten zurück geschrieben werden sollen. Dies wird in der Regel die gleiche Stelle sein aus der die Daten extrahiert wurden. Handelt es sich um relationale Daten, so ist es nicht trivial einen entsprechenden SQL-Befehl zum Update der Daten herzuleiten. In beiden Fällen sind zusätzliche Update-Informationen notwendig, wenn sich das Ziel der geänderten Daten von ihrer Herkunft unterscheidet. Aus diesen Gründen werden im Data Extraction/Update Knoten zusätzliche Informationen für diese Aufgabe vorgesehen. Die tatsächliche Ausführung dieser Extraktions- bzw. Update-Befehle wird von der Datenbank-Komponente (DB-Component) vorgenommen. Sie ist in der Lage die datenquellenspezifischen Extraktions-/Update-Befehle zu verarbeiten. Alle diese Knoten sind Nachfolger eines ausgezeichneten DataRoot Knoten. Er bildet die Wurzel des gesamten DataGraphs.

5.2.11.2.2. Datenstruktur

Die Darstellung der Datenstruktur im DataGraph ist für die Informationsvisualisierung ebenso wichtig, wie die reinen Datenwerte. Datenobjekte werden im DataGraph als eine ansonsten ungeordnete Menge von Schlüssel-Wert-Paaren dargestellt. Diese Schlüssel-Wert-Paare stellen die atomaren Daten im Datenmodell dar. Die Knoten, welche diese Mengen repräsentieren heißen DataRecord Knoten. Der Schlüssel stellt abstrakt eine eindeutige Bezeichnung für den zugeordneten Wert dar, d.h. mit dem Schlüssel kann eindeutig auf den Datenwert zugegriffen werden. Der zweite Knotentyp der im Modell der Datenstruktur verwendet wird, ist der sog. DataRecord-Set Knoten. Er stellt eine Menge von DataRecord Knoten dar. Die beiden Knotentypen werden nun in einem Graphen angeordnet, der die zugrunde liegende Datenstruktur repräsentiert. DataRecord Knoten können durch einen beliebigen Graphen verbunden werden. DataRecord-Set Knoten erhalten die in ihnen enthaltenen DataRecord Objekte als Nachfolger.

Die hierarchische Struktur eines XML-Dokumentes kann mit diesem Modell leicht abgebildet werden. Jedes Element eines XML-Dokumentes besitzt eine Menge von Attributen und steht in Vorgängerrelation zu einer Menge von weiteren Elementen. Ein Element kann aber auch lediglich Text enthalten. Attribute eines XML-Dokumentes sind nichts anderes als Schlüssel-Wert Paare, deswegen kann ein XML-Element als DataRecord Knoten dargestellt werden. Der Textinhalt eines XML-Elementes kann als ausgezeichnetes Attribut des DataRecord Knoten abgebildet werden. Die Vorgängerrelation kann durch die Graphenstruktur der DataRecord Knoten dargestellt werden. In der Abbildung 22 wird ein Baum aus DataRecord Knoten dargestellt, der ein XML-Dokument repräsentieren könnte. Grundsätzlich bietet *INVENTORY 3D* also die Möglichkeit auf XML als Datengrundlage zurückzugreifen, zurzeit ist jedoch lediglich der Zugriff auf Datenbanken realisiert.

Greift man auf eine relationale Datenbank zu, so ist das Ergebnis einer Anfrage eine Menge von Einträgen, sprich eine Tabelle. Auch wenn man dabei auf mehrere Tabellen zugreift, wird das Ergebnis der Anfrage im Rahmen einer Join-Operation immer als eine Tabelle dargestellt. Eine Tabelle besteht aus einer Menge von Spalten und Zeilen, wobei jede Spalte einen eindeutigen Bezeichner besitzt. Eine Tabellenzeile kann also als eine Menge von Schlüssel-Wert Paaren aufgefasst werden, wobei die Schlüssel die jeweiligen Spaltenbezeichner sind. Eine solche Tabellenzeile kann also als DataRecord Knoten dargestellt werden. Die Ergebnismenge einer Anfrage an eine relationale Datenbank kann wiederum durch einen DataRecord-Set Knoten mit den darin enthaltenen DataRecord Knoten repräsentiert werden. Die Struktur ist hier deshalb so einfach abzubilden, da bei der Extraktion von Daten aus einer relationalen Datenbank Schlüsselbeziehungen durch das Zusammenfassen in einer Tabelle wegfallen, die resultierende Tabelle aber noch alle Verbindungen

ausdrückt. Wollte man die ursprüngliche Struktur der Daten abbilden, bräuchte man weitere Knotentypen.

5.2.11.2.3. Abbildungsstruktur

In der Abbildungsstruktur soll dargestellt werden, wie die Daten aus der Datenstruktur mit der visuellen Struktur des Szenengraphs zusammenhängen. Der Zweck dieser Verbindung besteht darin, dass bei einer Änderung der Daten die Darstellung aktualisiert werden kann, ohne dass die gesamte Szene neu berechnet werden muss. Hierbei ist es von Bedeutung die Performanz bei der Visualisierung von Daten in Echtzeit zu erwähnen, aber auch in anderen Kontexten macht eine solche Aktualisierung Sinn. In einem Client-Server Szenario ist es häufig nicht praktikabel bei kleinen Änderungen ganze Szenen über das Netz zu Übertragen, in einem Multiuser-System wäre sogar eine ganze Reihe von Clients betroffen. Eine Abbildungsstruktur, die Zusammenhänge zwischen Datenstruktur und der visuellen Struktur ausdrückt, muss Knoten besitzen, die die Regeln beschreiben, aufgrund derer die Präsentationskomponenten instanziiert wurden. Eine solche Regel könnte beispielsweise die Form besitzen „Für jeden Nachfolger eines Knotens A, wird eine Instanz der Präsentationskomponente P erzeugt und nach dem Algorithmus X in die Szene eingefügt". Es sind aber auch ganz andere Regeln denkbar die zur Instanziierung von Präsentationskomponenten führen. Ein solches umfassendes Regelsystem zu entwickeln ist nicht Teil dieser Arbeit und soll hier nur erwähnt werden. Aus diesem Grund soll die vorzustellende Abbildungsstruktur auf die Informationswertvisualisierung beschränkt werden. Die Anforderung besteht nun darin, zum einen die Verknüpfung zwischen Daten und Attributen der Präsentationskomponenten zu vermerken und zum anderen vorher eine Transformation der Daten zuzulassen. Eine Transformation ist notwendig, wenn mit den extrahierten Daten vor der Visualisierung noch Berechnungen durchzuführen sind oder wenn die Daten zur korrekten Darstellung umformatiert werden müssen. Die Transformation der Daten wird durch die DataMap Knoten repräsentiert. Diese Knoten enthalten einen Algorithmus der eine Menge von Parametern erwartet und eine Menge von Ausgaben liefert. Jeder Parameterwert stammt entweder aus einem DataRecord Knoten oder aus dem Ergebnis eines anderen DataMap Knoten. Die Ergebnisse werden dann entweder auf das Attribut eines Präsentationsobjektes abgebildet oder dienen wiederum als Parameter für einen weiteren DataMap Knoten.

5.3. Implementierung

5.3.1. Verwendete Technologien

5.3.1.1. .NET-Framework von Microsoft

Das Microsoft .NET-Framework wurde für die Implementierung der Komponenten verwendet. Bei .NET handelt es sich um eine Entwicklungsplattform, die eine Reihe von Features besitzt, welche zum Teil erstmals in dieser Form zur Verfügung stehen. .NET unterstützt eine Reihe bekannter Programmiersprachen wie beispielsweise C++, Java (Microsoft) und Visual Basic, aber auch die im Rahmen der .NET Plattform neu entwickelte Sprache C#. .NET lässt sich am ehesten mit der Java-Plattform von SUN Microsystems vergleichen. Diese Gegenüberstellung wird zusätzlich zur Beschreibung der .NET Architektur in den nächsten Abschnitten vorgenommen, sofern dies sinnvoll ist.

5.3.1.1.1. Microsoft Intermediate Language und Metadaten

Alle im Rahmen des .NET Frameworks eingesetzten Programmiersprachen werden nicht direkt in den Maschinencode der gewünschten Zielplattform übersetzt sondern in eine

plattformunabhängige Zwischensprache, die Microsoft Intermediate Language[5]. MSIL ist eine bytecodierte Sprache für eine virtuelle Zielplattform. Um den MSIL Code auszuführen wird dieser von einem so genannten Just-in-Time[6] Compiler in den Maschinencode der Zielplattform übersetzt. Ein JIT-Compiler wird erst kurz vor der Ausführung des Codes angewendet, sobald das entsprechende Programm gestartet wird, bzw. sobald eine Programmbibliothek geladen wird. Das Kompilat geht nach Beendigung des Programms wieder verloren, d.h. dieser Prozess wird bei jedem Programmstart wiederholt. Java verfolgt einen ähnlichen Ansatz, wobei hier der Zwischencode[7] von der Java Virtual Machine ausgeführt wird. Diese Virtual Machine emuliert die virtuelle Plattform auf der Zielmaschine. Der Einsatz eines JIT Compilers ist unter Java hingegen optional. Der MSIL Code wird zusammen mit einer Reihe von Metadaten in eine so genannte Assembly verpackt. Darin sind alle Informationen über verwendete Klassen, Klassenstruktur, Methoden und andere verwendete Assemblies des Programms bzw. der Bibliothek enthalten.

5.3.1.1.2. Common Language Runtime

Die Common Language Runtime[8] stellt die Ausführungsumgebung für eine .NET Assembly dar. Sie enthält unter anderem den JIT Compiler der oben bereits erwähnt wurde. Sie basiert auf dem Common Type System[9] und der Common Language Specification[10] und besteht aus der Base Class Library und der .NET Execution Engine.

Das CTS beschreibt die grundlegenden Regeln für die Definition von Typen (Klassen, Interfaces, Structs, Einfache/Atomare Typen…) die von der CLR verarbeitet werden können. Es werden das Metadatenformat der Assemblies und die syntaktischen Konstrukte zur Typendefinition festgelegt. Das CTS umfasst alle Konstrukte zur Typendefinition die von einer .NET Programmiersprache verwendet werden können. Die CLS besteht aus einer Teilmenge der CTS und umfasst lediglich jene Konstrukte, welche von jeder .NET Sprache verarbeitet werden können. Alle Typen die außerhalb einer Assembly verwendet werden können müssen der CLS genügen.

Die Base Class Library ist eine Klassenbibliothek, die von der CLR zur Verfügung gestellt wird und damit in jeder .NET Sprache zu benutzen ist. Die darin vorhandenen Klassen stellen wesentliche Systemfunktionalität und weitere Dienste zur Verfügung. Beispiele für Dienste der Base Class Library sind IO, Threading und GUI.

Die .NET Execution Engine führt eine Assembly aus. Dazu werden die Metainformationen aus dem Assembly extrahiert und alle weiteren Assemblies/Klassen geladen, die zur Ausführung notwendig sind. Der JIT übersetzt die IL Instruktionen und führt den Code aus. Die .NET Execution Engine stellt nach dem Vorbild von Java einen Garbage Collector zur Verfügung, der automatisch unbenutzten Speicher freigibt.

5.3.1.1.3. .NET Programmiersprachen und Interoperabilität

Die Architektur des .NET Frameworks kann nun von beliebigen Programmiersprachen verwendet werden. Der Compiler dieser Programmiersprache muss IL Code bzw. .NET Assemblies erzeugen können. Außerdem muss die Sprache die syntaktischen Konstrukte und Typen unterstützen, die von der CLS gefordert sind und kann alle Features des CTS

[5] MSIL oder IL
[6] JIT
[7] sog. Java Bytecode
[8] CLR
[9] CTS
[10] CLS

verwenden. Mit diesen Forderungen ergibt sich eine vollkommene Interoperabilität zwischen allen .NET Programmiersprachen. Die erzeugte Assembly der einen Sprache kann dank der Metadaten und der CLS Spezifikation von jeder anderen Sprache verwendet werden. Durch die Metadaten sind alle Typinformationen von Klassen in der Assembly vorhanden. So kann eine Klasse aus einer Assembly in jeder .NET Sprache benutzt werden, als ob sie in der eingesetzten Sprache selbst programmiert worden wäre. Jede Klasse einer Assembly, die zur externen Verwendung vorgesehen ist, kann damit systemweit als Komponente verwendet werden. Diese kann nicht nur intanziiert werden, sondern auch als Basisklasse für weitere Vererbungsschritte dienen. Auch die Base Class Library wird so von allen Programmiersprachen gleich verwendet. Damit unterscheiden sich die verschiedenen .NET Sprachen nur durch ihre syntaktischen Konstrukte; der Zugriff auf die Systembibliotheken wie IO oder Threading wird immer über dieselben Klassen abgewickelt. Dieses Vorgehen unterscheidet sich sehr stark von der Java Plattform. Die Interoperabilität mit anderen Komponenten des Systems ist in Java nur schwer möglich.

5.3.1.1.4. Plattformunabhängigkeit

Ziel der Entwicklung von Java war eine komplette Plattformunabhängigkeit. Eine Implementierung der Java Virtual Machine ist ausreichend um Java Programme auf einer bestimmten Plattform auszuführen. Obwohl das Konzept von .NET auch eine plattformunabhängige Umgebung beschreibt (IL und CTS), wurde .NET zunächst mit Microsoft Windows als Zielplattform entwickelt. Wesentliche Teile von .NET wurden als Standard bei der ECMA verabschiedet [20]. Dieser Standard umfasst die Schnittstellen der CLR, die CLT, CLS und IL und wird als Common Language Intfrastructure[11] bezeichnet. Nicht im Standard enthalten ist jedoch die Base Class Library. Damit .NET Assemblies auf zwei verschiedenen Plattformen lauffähig sind, müssen diese jedoch exakt dieselben Schnittstellen der Base Class Library zur Verfügung gestellt bekommen. Das Problem hierbei ist, dass die fehlende Standardisierung spätere Änderungen möglich macht und dass eine komplette Spezifikation der Schnittstellen nicht zwingend von Microsoft bereitgestellt werden muss. Es gibt allerdings bereits Projekte .NET Compiler auf anderen Betriebssystemen zu implementieren. Das Mono Projekt beschäftigt sich mit einer Open Source Implementierung von C#, der CLR und der Base Class Library unter Linux [21]. Ob .NET in Zukunft plattformübergreifend unterstützt wird, ist im Moment noch nicht vorherzusagen.

5.3.1.1.5. Die C# Programmiersprache

Bei C# handelt es sich um eine von Microsoft im Rahmen des .NET Frameworks neu entwickelte Programmiersprache. Der syntaktische Aufbau ist sehr an Java und C++ angelehnt. Die Unterschiede zwischen C# und Java sind im Wesentlichen durch die besonderen Eigenschaften des .NET Frameworks begründet. So gibt es in C# keine Unterscheidung zwischen mehr zwischen atomaren oder einfachen Typen und Klassen, was bereits durch die CLS festgelegt ist.

5.3.1.1.6. Datenzugriff

Bei dem Zugriff auf Daten werden in .NET zwei Typen von Datenquellen favorisiert: XML Dokumente und relationale Datenbanken.

Der DOM Zugriff auf XML Dokumente ist fundamentaler Teil der .NET Plattform und ist bereits in der Base Class Library enthalten.

[11] CLI

Der Zugriff auf Daten aus relationalen Datenbanken[12] wird über die ADO.NET Klassenbibliothek abgewickelt, welche ebenfalls Teil der Base Class Library ist. ADO.NET stellt die Verbindung zu einer relationalen Datenbank über einen so genannten Data Provider her. Dieser nimmt die SQL Befehle entgegen und setzt das Ergebnis von Queries in das ADO.NET Objektmodell um. Dieses bildet die Tabellenstruktur als Klassen der A-DO.NET Bibliothek im Speicher nach und ermöglicht so einen sehr intuitiven Zugriff auf die Daten. Daten werden in dieser Tabelle geändert und über den entsprechenden SQL Befehl auch wieder von dort zurück geschrieben. Data Provider sind datenbank-/treiberspezifisch implementiert und zurzeit für Microsoft SQL Server, Oracle und ODBC verfügbar.

5.3.1.1.7. Weitere .NET Features

Ein wichtiges Feature des .NET Frameworks ist die sog. Reflection. Wie bereits erwähnt, extrahiert die CLR die Metadaten der geladenen Assemblies und führt den zugehörigen IL Code aus. Auf diese Metadaten kann nun durch einen speziellen Teil der Base Class Library zugegriffen werden: die Reflection. Mit den dort enthaltenen Klassen lassen sich alle Informationen über die verwendeten Klassen zur Laufzeit bestimmen, es lässt sich beispielsweise herausfinden welche Parameter eine Methode erwartet von der zunächst nur der Name bekannt ist. Außerdem lassen sich über die Reflection beispielsweise Typen erzeugen, von denen nur der Name bekannt ist (sofern dieser Typ in der Assembly enthalten ist).

Web Services sind ein weiteres Feature des .NET Frameworks. Ein Web Service ist eine besondere Klasse, deren Funktionalität über das Internet oder das lokale Netzwerk verwendet werden kann. Im Gegensatz zu anderen verteilten Komponentensystemen (DCOM/CORBA) wird hierbei jegliche Kommunikation im XML Format über Http[13] abgewickelt. Die Web Service Komponente wird auf einem Web Server installiert, der .NET Web Services unterstützt. Zur Benutzung des Web Services werden die XML Anwendungen WSDL und SOAP benutzt. Die WSDL[14] Datei eines Web Services enthält die plattformunabhängige Beschreibung aller Dienste bzw. Methoden, die dieser zur Verfügung stellt. Aufgrund der Beschreibung kann ein Client einen Proxy für diesen Web Service generieren, der einen transparenten Zugriff gestattet. Das Protokoll, welches zum Aufruf einzelner Methoden des Web Services verwendet wird, ist SOAP[15]. Es handelt sich um ein XML basiertes RPC Protokoll, welches vom W3C standardisiert wurde. Für den Client ist keine .NET Integration erforderlich. Dieser muss lediglich SOAP implementiert haben und sollte WSDL zur einfachen Client-Server Bindung verarbeiten können. SOAP lässt sich im Rahmen des .NET Frameworks nicht nur für Web Services, sondern ganz allgemein für RPC[16] Kommunikation in Client-Server Architekturen einsetzen. Eine bestehende Kommunikation zwischen Komponenten durch das SOAP Protokoll ist in .NET ohne großen Aufwand lösbar. Es muss nur sichergestellt sein, dass die Typen der übergebenen Parameter serialisierbar sind. Die Serialisierung wird von der CLR automatisch vorgenommen, sofern die zu serialisierenden Klassen mit einer speziellen Markierung versehen sind.

Das .NET Framework ermöglicht es bestehende Komponenten der älteren COM[17] Technologie von Microsoft in neue .NET Anwendungen zu integrieren. Unter Microsoft Windows war COM der bisherige Standard für Software Komponenten. .NET importiert sol-

[12] RDBMS
[13] Hypertext Transfer Protocol
[14] Web Service Description Language
[15] Simple Object Access Protocol
[16] Remote Procedure Calls
[17] Component Object Model

che Komponenten, indem sie durch eine Proxy-Klasse gekapselt werden. Dieser Proxy ermöglicht den Zugriff auf das COM Objekt, als ob es sich um ein .NET Objekt handelt. Die Integration verläuft damit in der Regel schnell und einfach.

5.3.1.1.8. Performanz

In [22] wird ein Überblick über die Problemfelder der Performanz des .NET Frameworks gegeben. Der Autor begründet nachvollziehbar, dass man unter .NET im Gegensatz zu herkömmlichen Compiler Technologien mit keinen Performance-Einbussen rechnen muss. Folgende Argumente sollen das herausstellen:

- Der JIT Compiler arbeitet inkrementell, das bedeutet, dass jeweils nur der Code übersetzt wird, der gerade verwendet wird. Einmal übersetzter Code wird nicht noch einmal übersetzt. Es kommt damit nicht zu langen Startzeiten.
- Optimierungen des JIT Compilers sind effizienter als die herkömmlicher Compiler, da dieser die verwendeten Optimierungen exakt an die verwendete Prozessorarchitektur anpassen kann.
- Der JIT Compiler ist effizienter als ein herkömmlicher Compiler, da er bestimmte Optimierungen zur Laufzeit anpassen kann. Die Ersetzung von Funktionsaufrufen durch Inline-Funktionen kann aufgrund des beobachteten Laufzeitverhaltens vorgenommen werden.
- Da dynamisch gebundene Bibliotheken ebenfalls vom JIT Compiler übersetzt werden müssen, kann auch bei deren Aufruf eine Inline-Optimierung stattfinden. Das ist bei herkömmlichen Compilern nur bei statischen Bibliotheken möglich.
- Die eingesetzte Garbage Collection ermöglicht eine schnellere Speicherverwaltung. Freier Speicher ist immer konsekutiv und dadurch sofort zu reservieren. Die Garbage Collection defragmentiert den Speicher nach der Freigabe.

5.3.1.1.9. .NET als Grundlage für die Implementierung

Es gibt einige Gründe, die für .NET als Plattform für die Implementierung gesprochen haben:

- Das .NET Framework implementiert eine flexible und fortschrittliche Komponententechnologie, welche die spätere Erweiterbarkeit dieser Anwendung sicherstellt. Darüber hinaus ist es jedem Entwickler freigestellt in welcher Programmiersprache er diese Komponenten einsetzt. Es können alle Methoden der Objektorientierung voll eingesetzt werden.
- Die Konzepte der .NET Assemblies, der Metadaten und der Reflection erweitern die Möglichkeiten der Implementierung erheblich.
- Die Erstellung von Client-Server Architekturen ist unter .NET dank Web-Services und SOAP mit einfachen Mitteln möglich. Die Verteilung bestehender Komponenten auf Client und Server ist leicht.
- Trotz der Verwendung von IL ist nicht mit signifikanten Einbußen der Performanz zu rechnen.
- .NET bietet sehr einfachen Zugang zu XML Daten und relationale Datenbanken.

- Die Integration bestehender COM Komponenten in .NET Anwendungen ist problemlos.
- .NET ist insgesamt eine fortschrittliche Plattform, die bereits eine wichtige Technologie in der Softwareentwicklung darstellt und mit großer Wahrscheinlichkeit in Zukunft eine noch signifikantere Rolle spielen wird.

5.3.1.1.10. .NET Speech

Die Anwendung und Benutzung von *INVENTORY 3D* soll so einfach wie möglich sein. Aus diesem Grund wurde das Man-Machine-Interface um die Spracheingabe erweitert. Die Firma Microsoft bietet im Rahmen eines Beta-Programms die Möglichkeit an ASP.NET Anwendungen mit Spracheingabe und Sprachausgabe (Text-To-Speech) zu erweitern. Hierbei kommen vorgefertigte ServerControls (serverseitig gebundene Steuerelemente) zum Einsatz, die die Spracheingabe ermöglichen und englische Sprache erkennen bzw. englischsprachige Texte ausgeben können. Die Verarbeitung der Textinformationen erfolgt auf dem Server und wird über die .NET-Speech-Erweiterung (Version Beta 3.0) des .NET-Frameworks gesteuert. Bei langsamen Internetverbindungen sind die Spracherkennung und die damit verbundenen Reaktionen des Servers ebenfalls langsam. Aus diesem Grund wurde nach einer Alternative gesucht, die die Spracherkennung innerhalb einer Desktopanwendung bereitstellt. Das .NET-Framework bietet keine Sprachunterstützung für Desktopanwendungen. Stattdessen muss man auf Library´s (DLL´s) der Speechengine zurückgreifen und diese über eine Wrapperklasse in die eigene Anwendung einbinden.

5.3.1.2. VRML Cortona Automation Interface (CAI) von Parallelgraphics

Der Cortona VRML Client ist eine von der Firma ParallelGraphics [23] entwickelte Komponente zur Darstellung von VRML Szenen. Der Cortona Client ist kostenlos verfügbar. Der Cortona Client wird als Viewer für die Darstellung der 3D-Szene in der Beispielanwendung *INVENTORY 3D* eingesetzt. Das Extrahieren einer VRML Beschreibung aus der SceneGraph Datenstruktur ist einfach und effizient durchführbar. Außerdem besitzt der Cortona Client eine Schnittstelle, die einen Zugriff auf die dargestellte Szene ermöglicht. Dieser Zugriff ist notwendig, um den Inhalt der Szene zur Laufzeit ändern zu können. Dabei gibt es allerdings Einschränkungen zu beachten, die nur durch umständliche und aufwendige Eigenentwicklungen kompensierbar sind. Der Cortona Client bietet außerdem die Möglichkeit Benutzereingaben in Form von Ereignissen abzufragen. Dies ist für die Implementierung des Modells der Benutzerinteraktion wichtig. Der Cortona Client liegt als COM Komponente vor und lässt sich damit problemlos in die .NET Umgebung integrieren.

5.3.2. Bemerkungen zur Implementierung

5.3.2.1. Verwendete Tools

Da bei der Implementierung von *INVENTORY 3D* mehrere Technologien zum Einsatz kamen, wurden unterschiedliche Werkzeuge eingesetzt, die die Arbeit erleichterten. Hierbei soll nur auf die wichtigsten Tools eingegangen werden. Zu erwähnen wäre auch das Microsoft Office Produkt Access 2003, das zur Konzeption der Artikeldatenbank diente und auch zu Demonstrationszwecken eine betriebsfähige SQL-Server Datenbank ersetzen sollte. Das verwendete Betriebssystem zur Entwicklung und zur Demonstration der Anwendung war Windows Server 2003 Web Edition. Besonders die Verwaltungssimplizität und

Zuverlässigkeit für die mobile ASP.NET Webanwendung favorisierten diesen neuen Server von Microsoft. Als Webserver diente der in Windows Server 2003 integrierte IIS 6.0[18].

5.3.2.1.1. Microsoft Visual Studio .NET 2003

Die Entwicklungsumgebung (IDE[19]) Visual Studio .NET bietet Entwicklern, die .NET-Anwendungen implementieren, umfangreiche Funktionen und integriert alle .NET-Technologien (ASP.NET, ADO.NET, mobile ASP.NET, WebServices, .NET Compact Framework, Windows Forms etc.). Damit war Visual Studio .NET die erste Wahl als IDE. Zusätzliche Pakete, wie z.B. Speech.NET, konnten problemlos installiert werden und wurden automatisch in Visual Studio .NET verankert. Auch das Nutzen von nicht verwaltetem Code[20], was beim Einsatz des Cortona Viewers und der Speechengine nötig war, wurde durch die IDE vereinfacht und ermöglichte eine komfortable Programmierung der Schnittstellen. Sehr hilfreich war die Debuggingfunktion für ASP.NET Anwendungen. In Verbindung mit dem IIS war ein debuggen in ähnlicher Weise möglich, wie Entwickler es von Desktopanwendungen gewohnt sind.

5.3.2.1.2. Openwave SDK 6.2.2

Das Openwave SDK 6.2.2 enthält einen Telefonemulator für Geräte mit Openwave Mobile Browser. Damit lassen sich sowohl Dienste testen, die XHTML mit CSS nutzen, als auch cHTML[21] und WML[22] Seiten anzeigen. Außerdem ist kein WAP Gateway nötig, da man den HTTP Direct Mode nutzen kann. Zusätzlich ist auch eine Verbindung zu einem "live" WAP Gateway möglich.

Für den realen Einsatz von *INVENTORY 3D* sind eindeutig PDA[23] zu bevorzugen, für die Demonstration reicht die Implementierung für einfache Handys aus. Zumal die Aufgabe der mobilen Clients lediglich das Aufnehmen und Aktualisieren der Artikeldaten beinhaltet und deswegen keine ausgefeilte Benutzeroberfläche notwendig ist. Aus diesem Grund wurde auf die Emulierung eines PDA-Betriebssystems wie z.B. Windows Mobile 2003 verzichtet und eine mobile Webanwendung für Handys mit nur sehr geringen Anforderungen an die Hardware entwickelt.

5.3.2.1.3. ParallelGraphics VrmlPad 2.0

VrmlPad von ParallelGraphics ist ein Tool, das ein erleichtertes Editieren und Erstellen von VRML-Dateien ermöglicht. Die Bedienung ist sehr intuitiv und erlaubt über die Darstellung in der Baumstruktur ein rasches Finden und Manipulieren gesuchter Knoten. So erfolgte die Generierung des Szenegraphen zuerst manuell und statisch mithilfe von VrmlPad und konnte daraufhin inkrementell durch die Konstrukte innerhalb der SceneGeneration-Klassen ersetzt werden, um dynamisch die Szene aufzubauen. Auch kleinere Verbesserungen an visuellen Elementen ließen sich gut mit VrmlPad testen und dann in der Anwendung implementieren.

[18] Internet Information Server
[19] integrated development environment
[20] unmanaged Code
[21] Compact Hypertext MarkUp Language
[22] Wireless MarkUp Language
[23] Personal Digital Assistent

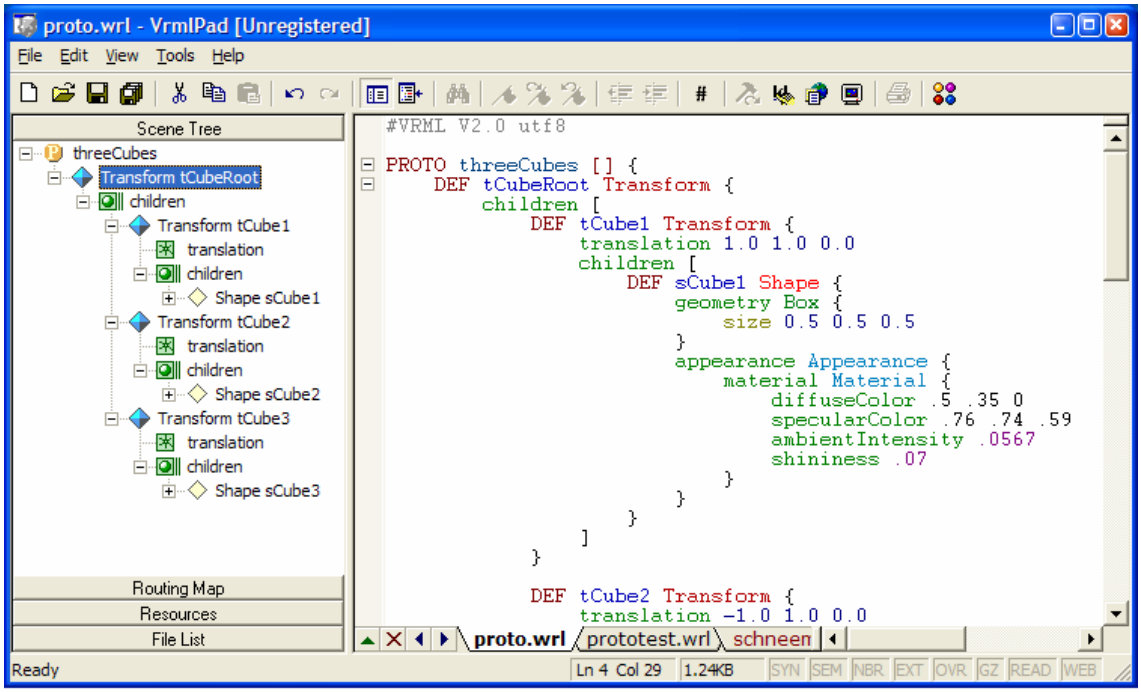

Abbildung 23: Der Vrml-Editor VrmlPad im Einsatz

5.3.2.1.4. NUnit

Da besonders die Klassen für die Szenengerierung ständig komplexer wurden, blieb es nicht aus in kurzen Abständen den Code zu refaktorieren. Refaktorieren ist ein Begriff aus der Softwaretechnik und soll hier nicht weiter erklärt werden. Man kann es kurz als ein Verbessern und effizienter Machen des Codes auffassen. Wichtig dabei ist, dass nach dem Refaktorieren die Funktionalität und somit die Richtigkeit und Konsistenz der Implementierung erhalten bleibt. Dazu dienen sog. Unittests, die man am besten noch vor der eigentlichen Implementierung konzipiert. Diese lassen sich sehr gut mit dem Tool NUnit, das ein C#-Klon von JUnit ist, bewerkstelligen und verwalten.

5.3.2.1.5. Microsoft Visio for Enterprise Architects

Microsoft Visio stellte sich als ein hilfreiches Tool zur Planung der Anwendung *INVENTORY 3D* heraus. Besonders bei der Konzeption des SceneManagers, der für die Szenengeneration verantwortlich ist, war die Erstellung von übersichtlichen UML-Diagrammen sehr effizient. Die erzeugten UML[24] Strukturen(siehe Abbildung 24) ließen sich dann auch problemlos in Code konvertieren und mit Funktionalität füllen.

[24] Unified Modelling Language

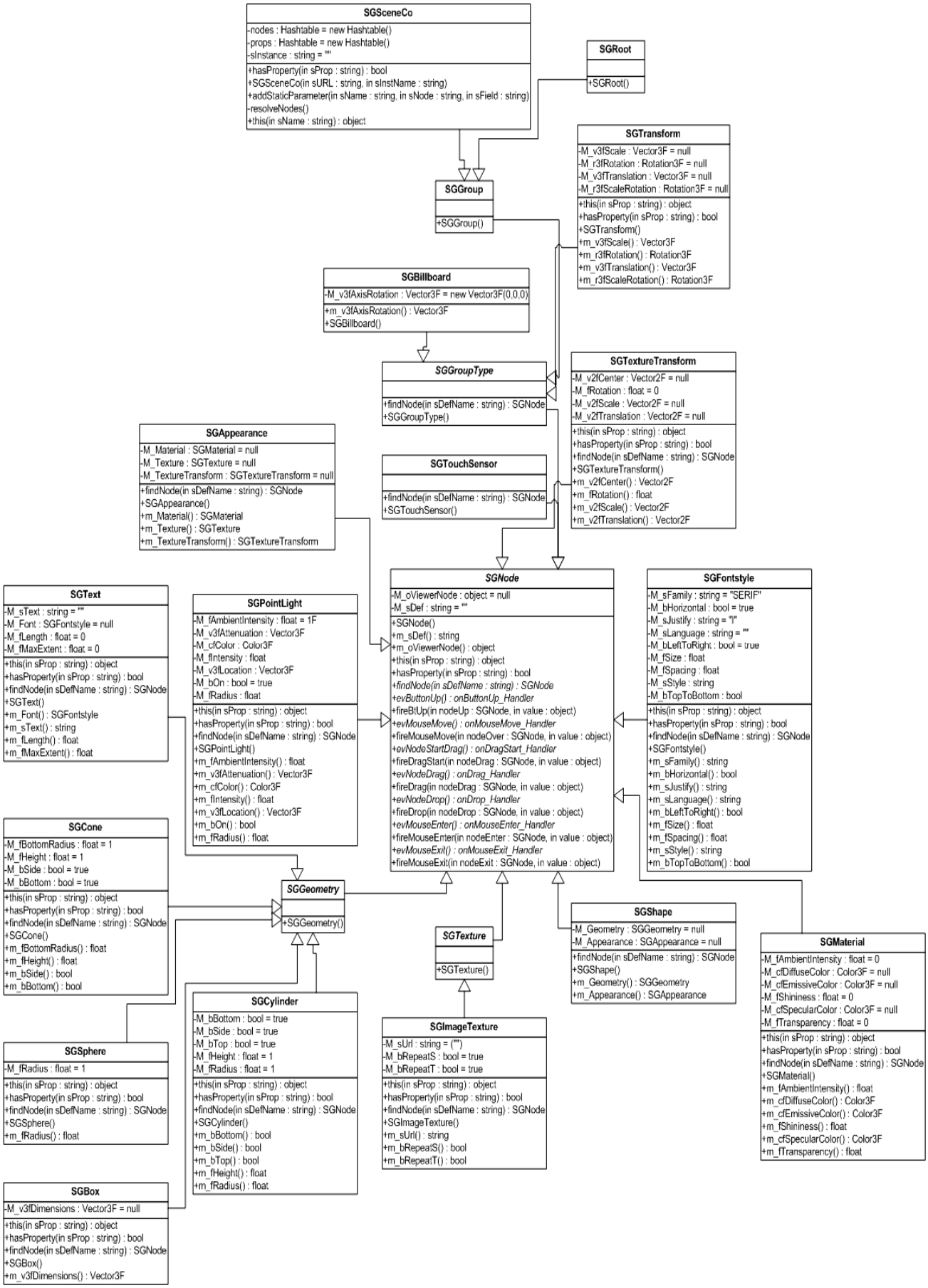

Abbildung 24: UML-Diagramm mit allen implementierten Szenengraphelementen

5.3.2.2. *INVENTORY 3D* Desktop Client

Der Desktop Client ist die Komponente von *INVENTORY 3D*, die für die Darstellung der Visualisierung verantwortlich ist. *INVENTORY 3D* ist eine Anwendung zur Visualisierung von Inventurdaten und hat damit als wichtigstes Datum den Artikel. Ein Artikel besitzt die Attribute Name, Artikelnummer, Menge, Reorderlevel, Gesamtwert und Einzelpreis. In einer realen Anwendung müsste die Attributmenge um weitere wichtige Eigenschaften erweitert werden. Die Attribute Menge und Reorderlevel sind durch Säulen visualisiert, wobei die Reordersäule transparent ist und die Artikelmengensäule umgibt. Ein Umschalten der Visualisierungsattribute auf Einzelpreis oder Gesamtwert ist ebenfalls möglich. Die Artikelsäulen sind in einem Kreis angeordnet, dessen Umfang von der Anzahl der unterschiedlichen Artikel abhängig ist. Der Kreis bietet zwar den Vorteil, dass theoretisch unendlich viele Säulenelemente angeordnet werden könnten, aber ein großer Umfang erzeugt auch eine große Kreisfläche, die noch ungenutzt ist. Diese zur Verfügung stehende Fläche kann den Ansatz für weiterführende Entwicklungen von *INVENTORY 3D* bilden, da man sie für Visualisierungselemente neuer Artikelattribute nutzen könnte. Zur Orientierung ist der Kreis mit einer Skala aus größer werdenden Einheitslinien versehen, deren zugehörige Einheit geändert werden kann, da beim Sortieren nach Namen, Artikelnummer, Reorderlevel, Gesamtwert oder Einzelpreis nicht ersichtlich ist, wo der kleinstwertigste Artikel bzw. der größtwertigste Artikel zu finden ist. Die Sortierfunktion ist sowohl über eine Combobox auswählbar, als auch über einen Sprachbefehl.

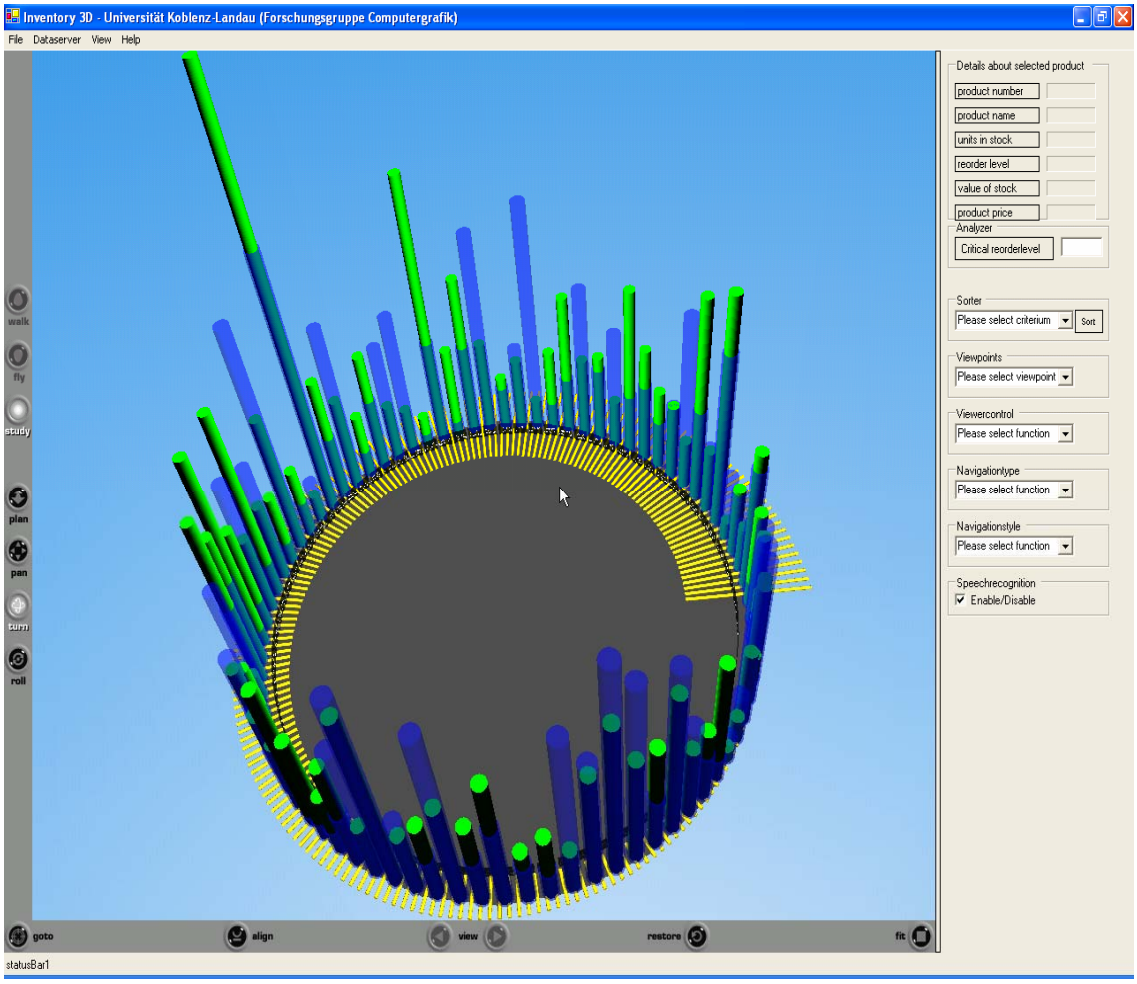

Abbildung 25: Darstellung des GUI des Desktop Clients

Bei den Sprachbefehlen gibt die Sprachengine über die Text-To-Speech-Funktion immer eine Bestätigung, wenn ein Befehl erkannt wurde. In Tests hat sich jedoch herausgestellt,

dass die Spracherkennung durchaus nicht unproblematisch ist. Umgebungsgeräusche wurden zum Teil als Befehle interpretiert, was aber auch auf ein minderwertiges Headset zurückzuführen war. Eine mögliche Lösung des Missinterpretationsproblems wäre das Einbauen eines Startbefehls, der zuerst erkannt werden muss, bevor der Funktionsbefehl diktiert werden kann. Dieser Ansatz funktionierte zwar, war aber doch sehr umständlich. Abschließend zur Spracherkennung in *INVENTORY 3D* lässt sich sagen, dass eine Bedienung über Sprachsteuerung da sinnvoll ist, wo keine Tastatur oder Maus eingesetzt werden kann. Am Tischarbeitsplatz sind jedoch die konventionellen Eingabegeräte zu bevorzugen, da die Umsetzung in Funktionsreaktionen der Anwendung wesentlich schneller und genauer erfolgt.

Die grafische Benutzeroberfläche ist aufgeteilt in die linke Visualisierungskomponente mit den Steuerelementen des Cortona Viewers und ein rechtes Bedienungspanel, das Detaildaten zum ausgewählten Artikel, Comboboxen zum Sortieren und Viewpoint ändern, und weitere Interaktionselemente enthält.

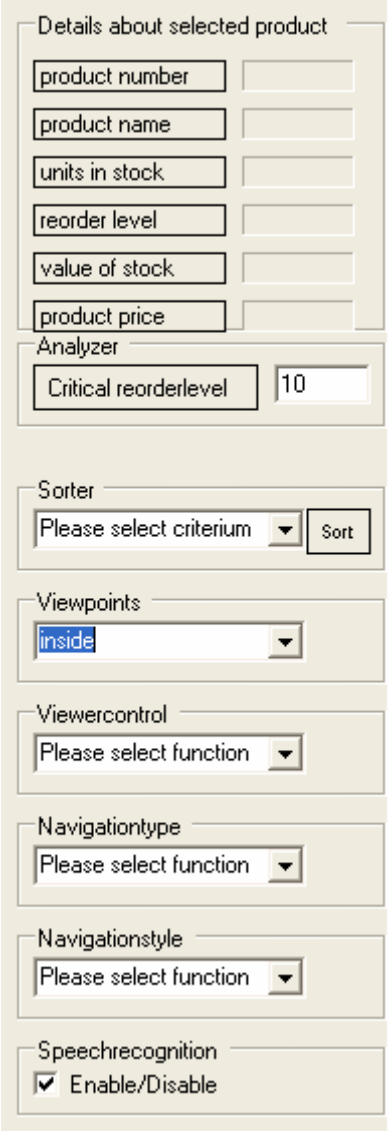

Abbildung 26: Bedienungspanel mit Analysefunktionen und Artikeldetails

Alle Navigationsfunktionen des Cortona Viewers sind sowohl über das Viewerfeld als auch über ausgelagerte Bedienungselemente zu erreichen. Diese Auslagerung ist durch bereitgestellte Schnittstellen unproblematisch umgesetzt und erlaubt auch die Interaktion mit

Funktionen aus Klassen der Sprachengine. Somit ließe die Anwendung theoretisch zu, dass alle Navigationsfunktionen über Sprache gesteuert werden könnten. Da dies jedoch zu ineffizient und unpraktikabel wäre, wurde auf eine Implementierung von Sprachbefehlen zur Steuerung der Viewerfunktionen verzichtet.

Es sind drei unterschiedliche Viewpoints vorprogrammiert, zwischen denen über Spracheingabe und manuelle Bedienung gewechselt werden kann. Um einen Überblick über alle Artikel zu bekommen, bietet sich der Viewpoint Overview an.

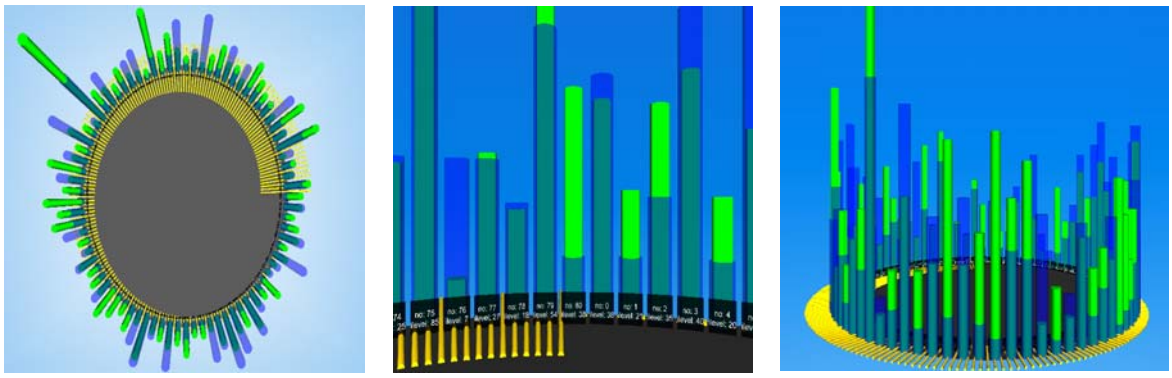

Abbildung 27: Viewpoints von links nach rechts: Overview, Inside, Outside

Der „Insideview" erlaubt ein Rotieren durch alle Artikel und somit eine Selektion des gewünschten Artikels. Der „Outsideview" ermöglicht wiederum einen Überblick und eine bessere Erfassung der Säulengrößen.

Als Analysefunktion ist das Festlegen eines Schwellwertes für den Reorderlevel beispielhaft implementiert worden. Dabei handelt es sich um eine wichtige Funktion, die es dem Lagerverwalter erlaubt die Artikel zu ermitteln, deren Lagerbestände in nächster Zeit neu aufgefüllt werden müssen. Weiterhin lässt sich über die Sortierfunktion der Wichtigkeitsgrad der Neubestellung ermitteln.

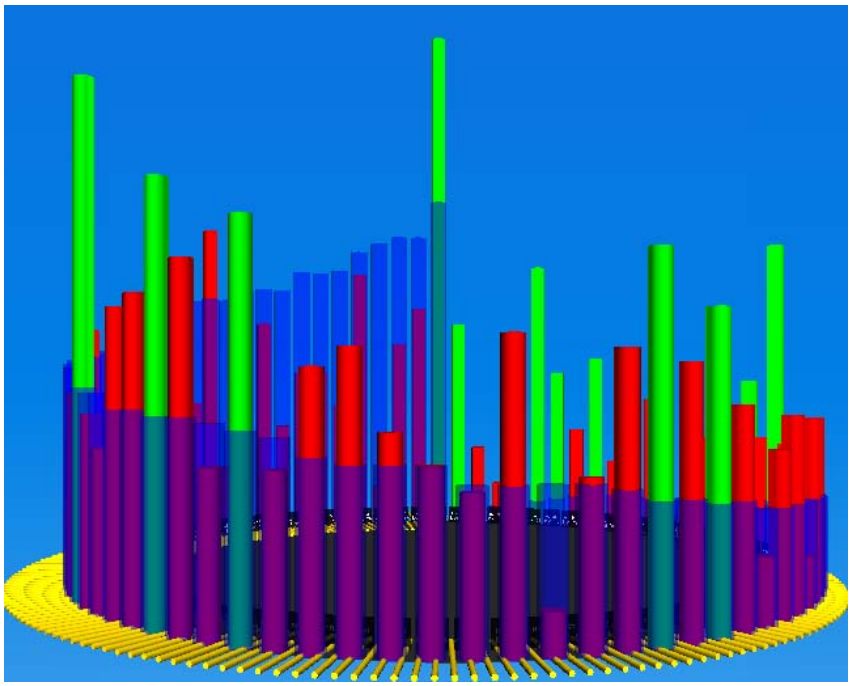

Abbildung 28: Visualisierung nach Ausführen der Reorderfunktion

6. Zusammenfassung und Ausblick

INVENTORY 3D erfüllt alle Anforderungen, die in der Aufgabenstellung gefordert waren. Inwieweit die vorhandene Visualisierung gelungen ist, wäre durch einen Test in einer realen Umgebung zu verifizieren. Auch die Spracheingabe wurde umgesetzt und zum Teil getestet. Hierbei lässt sich sagen, dass Spracheingabe durchaus ein hilfreiches Feature sein kann, aber die konventionellen Eingabemittel noch zu bevorzugen sind. Die Umsetzung von *INVENTORY 3D* nach Client/Server-Templates war sehr hilfreich und der einzig praktikable Weg eine verteilte Anwendung zu erzeugen. In weiterführenden Entwicklungen wäre ein Umstieg vom Smartphone oder Handy zum PDA notwendig, da auch eine Visualsierung für die „Thin Clients" denkbar ist und deswegen ein größeres Display und mehr Prozessorleistung erfordern würde. Der Einsatz des .NET-Frameworks hat sich als richtige Entscheidung herausgestellt, da alle verwendeten Technologien im Rahmen des .NET-Frameworks integriert und mit der bevorzugten IDE Visual Studio .NET implementiert werden konnten. So wäre z.B. der Einsatz der Programmiersprache Java in vielen Bereichen komplizierter ausgefallen. Eine Spracherkennung im Rahmen einer Java-Anwendung oder das Einbinden von mobilen Clients in eine Client/Server-Struktur hätte vermutlich mehr Zeit beansprucht als dies bei der vorhandenen Anwendung der Fall war.

Unterschätzt wurde allerdings der Aufwand für die Konzeption des Szenegraphen und die dafür verantwortliche Klasse SceneManager. Der erste Ansatz sah lediglich einen Zugriff und eine Manipulation der visuellen Elemente auf VRML-Ebene vor, was aber schließlich durch einen vollständig dynamisch erzeugten Szenegraphen ersetzt wurde. Hierbei machten viele Aspekte der objektorientierten Programmierung einen Großteil der Refactoring-Arbeit aus, die nach einiger Zeit notwendig wurde. Besonders der „event/delegate-Mechanismus" erlaubt es, dass die Szenegraphobjekte elegant miteinander Nachrichten austauschen, wodurch eine Aktualisierung von Werten erfolgen kann. Die Szenengraphkomponente von *INVENTORY 3D* ist soweit getestet, dass sie in anderen Projekten ihren Einsatz finden könnte. Zwar sind nicht alle VRML-Elemente implementiert, aber durch den Vererbungs- und Interfacemechanismus lassen sich weitere Klassen anlegen, die zukünftige Szenegraphen flexibler und mächtiger machen könnten.

Im Rahmen dieser Studienarbeit wurde die Anwendung *INVENTORY 3D* vor Studenten und Betreuern vorgestellt, um eine Resonanz und Anregungen von anderen Entwicklern als dem Autor zu erhalten. Die anwesenden, angehenden Computervisualisten und Informatiker bewiesen Ideenreichtum und Phantasie bei der Schöpfung neuer visueller Elemente, die in einer Lagerverwaltung einen Platz finden könnten. So wurde z.B. der Vorschlag gemacht, dass transparente Tafeln mit Artikeldetaildaten per Hooverfunktion[25] über die Säulen geblendet werden könnten. Auch der Einsatz einer hierarchischen Visualisierung

[25] Element wird dargestellt, sobald der Zeiger sich über dem festgelegten Objekt befindet.

wurde genannt. Hierbei könnten ausgewählte Artikelsäulen mehrere Artikel zu Gruppen vereinen. Bei Selektion einer solchen hierarchisch übergeordneten Säule würde sich dann ein neuer Kreis von Artikeln öffnen, die sich in der ausgewählten Gruppe befinden. Problematisch wäre jedoch, dass man bei sehr vielen Hierarchieebenen und Artikeln nicht mehr alle Artikel simultan darstellen könnte. Interessant fand das Auditorium auch den Einsatz der Spracheingabe von Befehlen zur Steuerung von *INVENTORY 3D*. Die Spracheingabe ist momentan begrenzt auf den Desktop Client und würde noch mehr Sinn machen, wenn sich auch die PDA per Spracheingabe steuern ließen. Diesbezüglich kann man erwähnen, dass für den Pocket Internet Explorer ein sog. Speech-Addin Version 1.0 existiert, dass Speech.NET-Controls interpretieren kann. Somit wäre die Spracherkennung und Sprachausgabe auf dem PDA in zukünftigen Versionen von *INVENTORY 3D* realisierbar und könnte auf der vorhandenen Konzeption aufsetzen.

Die aktuelle Version von *INVENTORY 3D* verwendet VRML für den Szenegraphen. Eine Alternative dazu wäre X3D. Der Wechsel von VRML zu X3D würde viele Vorteile und Verbesserungsansätze für die derzeitige Anwendung mit sich bringen. So bietet das .NET-Framework die Möglichkeit Klassen (bzw. Objekte) in XML zu serialisieren. Dadurch könnte man theoretisch den gesamten Szenegraphen in das X3D-Format serialisieren und aus einer X3D-Datei einen programminternen Szenegraphen erzeugen (deserialisieren). Voraussetzung dafür ist jedoch ein schneller X3D-Renderer, der zur Entwicklungszeit dieser Studienarbeit nicht zur Verfügung stand.

Literaturverzeichnis

[1] Dässler, Rolf: „Informationsvisualisierung: Stand, Kritik, Perspektiven in Methoden/Strategien der Visualisierung in Medien, Wissenschaft und Kunst“, Wissenschaftlicher Verlag Trier (WVT) 1999.

[2] R.Däßler und H.Palm: „Virtuelle Informationsräume mit VRML.“ dpunkt Verlag, Heidelberg 1998.

[3] XEROX PARC, User Interface Research Group, http://www.parc.xerox.com

[4] N.Gershon, S.G.Eick und S.Card: “Information Visualization”, interactions 3/4 (1998), 9-15

[5] Medialab, University of Georgia

[6] Card, Mackinlay und Shneidermann.: „Readings in Information Visualization“, Morgan Kaufmann Publishers, San Francisco 1999

[7] Shneiderman und Ben: „The eyes have it: A task by data type taxonomie for information visualizations”, Visual Languages, 1996.

[8] Killenberg, H.; Kuhlen, R.; Manecke: „Wissensbasierte Informationssysteme und Informationsmanagement“, S. 26, 39. Universitätsverlag Konstanz. 1991

[9] Meghini, C.; Sebastiani, F.; Straccia, U.; Thanos, C.: “A Model of Information Retrieval Based on a Terminological Logic.” S. 298, 308. 1993

[10] Dachselt und Ebert: „Collapsible Cylindrical Trees: A Fast Hierarchical Navigation Technique“, Proceedings of the IEEE Symposium in Information Visualization 2001 (INFOVIS’01)

[11] Rekimoto, J., Green, M.: “The Information Cube: Using Transparency in 3D” Dezember 1993

[12] Muir, E.; O'Neill, R.: “Perspective-based Interface using an extended Masthead”, U.S.-Patent 5621906. 1997

[13] Mackinlay, Jock D.; Georg G. Robertson, Stuart K. Card: “The Perspective Wall: Detail and Context Smoothly Integrated” p. 173-197., 1991

[14] http://www.cartia.com

[15] http://www.sgi.com

[16] Robertson, Cameron, Czerwinski, Robbins: „Visualizing Patterns: Polyarchy visualization“, Proceedings of the SIGCHI conference on Human factors in computing systems, IEEE 2002

[17] Schönhage, von Ballogooij, Elliens: „3D gadgets for business process visualization – a case study“, Proceedings of the Web3D-VRML 2000 fifth symposium on Virtual reality modeling language“, ACM Press, New York 2000

[18] Chi, Pitkow, Mackinlay, Pirolli, Gossweiler, Card: „Visualizing the Evolution of Web Ecologies“, Conference proceedings on Human factors in computing systems, ACM Press 1998

[19] Web3D Consortium: „VRML97 Specification ISO/IEC 14772-1:1007“, http://www.web3d.org/technicalinfo/specifications/vrml97/index.htm, 1997

[20] Standard ECMA-335 „Common Language Infrastructure (CLI) Partitions I-IV“, http://www.ecma.ch, 2001

[21] Homepage des Mono Projektes, www.go-mono.com, 2004

[22] Emmanuel Schanzer: „Performance Considerations for Run-Time Technologies in the .NET Framework, A .NET Developer Platform White Paper“, http://www.msdn.microsoft.com, 2001

[23] Parallel Graphics Homepage, Cortona VRML Client, www.parallelgraphics.com , 2004