Context-Sensitive Visualization

Vom Fachbereich Informatik der Universität Kaiserslautern
zur Erlangung des akademischen Grades
Doktor der Naturwissenschaften (Dr. rer. nat.)
genehmigte Dissertation
von

Dipl.-Inform. Achim Ebert

Dekan
Prof. Dr. Hans Hagen

Promotionskommission

Vorsitz	Prof. Dr. Otto Mayer
1. Berichterstatter	Prof. Dr. Hans Hagen
2. Berichterstatter	Prof. Dr. Paul Müller

Mündliche Aussprache: 15. April 2004

D386

Meiner Familie

Danksagung

Als Erstes möchte ich meinem Doktorvater Prof. Dr. Hans Hagen dafür danken, dass er mir die Möglichkeit geboten hat, diese Arbeit als Mitglied seines Forschungsbereiches IVS (Intelligente Visualisierung und Simulation) an der Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI) GmbH anzufertigen. Besonders danke ich ihm für die vielen Hinweise, Anregungen und Diskussionen, die einen entscheidenden Einfluss auf die entstandene Arbeit hatten. Vor allem möchte ich mich aber auch für die Gelegenheit, zahlreiche nationale und internationale Workshops und Konferenzen zu besuchen, recht herzlich bedanken!

Ein herzliches Dankeschön geht an die Studentinnen und Studenten, sowie Kolleginnen und Kollegen, deren wertvolle Beiträge die Entstehung der vorliegenden Arbeit in dieser Form erst ermöglicht haben. Insbesondere möchte ich an dieser Stelle (alphabetisch geordnet) namentlich folgende Personen nennen: Henning Barthel, Andreas Divivier, Jochen Ehret, Ingo Ginkel, Juri Schädlich und Dennis Wagner. Ein besonderer Dank geht an dieser Stelle an Michael Bender, der die Entstehung dieser Arbeit durch zahlreiche wissenschaftliche Diskussionen und Hinweise entscheidend unterstützt hat. Auch trägt natürlich ein gutes Arbeitsklima maßgeblich zum Gelingen einer solchen Arbeit bei – ein Dankeschön geht daher an dieser Stelle auch an Mady Gruys und Roger Daneker.

Meiner Familie und meinem Freundeskreis danke ich vor allem für ihr Verständnis und ihre Unterstützung in der häufig auch an Abenden und Wochenenden doch recht arbeitsreichen und entsprechend freizeitarmen Zeit.

Zusammenfassung

Im Informationszeitalter haben die Menschen überall und jederzeit Zugang zu einer kontinuierlich ansteigenden Fülle von Informationen. Hierzu trägt vor allem die explosionsartig wachsende globale Vernetzung der Welt, insbesondere das Internet, maßgeblich bei. Die Transformation der verfügbaren Informationen in Wissen sowie die effiziente Nutzung dieses Wissens stellen dabei entscheidende Faktoren für den Erfolg eines Unternehmens oder eines Einzelnen dar.

Es stellt sich also die Frage: *Leben wir im Informationszeitalter?* Diese Frage erinnert an die von Immanuel Kant in [65] gestellte Frage „*Leben wir jetzt in einem aufgeklärten Zeitalter?*“ und dessen Antwort „*Nein, aber wohl in einem Zeitalter der Aufklärung.*“. Entsprechend lässt sich auch die Frage „*Leben wir in einem informierten Zeitalter?*“ mit „*Nein, aber wohl in einem Zeitalter der Information*“ beantworten (vergleiche [14]).

Das Problem, dass sich die überwältigende Fülle an Information ohne geeignete Hilfsmittel vom Menschen nicht oder nur schwer beherrschen lässt, hat im Laufe des letzten Jahrzehnts maßgeblich zur Entwicklung des äußerst dynamischen Forschungs- und Anwendungsgebietes der Visualisierung als Teilgebiet der Computergrafik beigetragen. Der Grund hierfür liegt in der Tatsache, dass der Mensch wesentlich besser mit visuellen Eindrücken als mit abstrakten Zahlen oder Fakten umgehen kann. Die Erkennung von Mustern in Daten (z. B. Gruppierungen und Häufungen) wird durch die Visualisierung stark vereinfacht und lässt vielmals Zusammenhänge zwischen Daten überhaupt erst greifbar werden.

Unter computergestützter Visualisierung versteht man die in der Regel interaktive grafische Umsetzung von Daten. Handelt es sich dabei um physikalische Daten (z. B. entstanden durch Messvorgänge), so spricht man von *Scientific Visualization*. Handelt es sich eher um abstrakte bzw. nicht-physikalische Daten, so ordnet man die entsprechenden Verfahren der *Information Visualization* zu. Beide Teilgebiete der Visualisierung verfolgen jedoch das gemeinsame Ziel, Informationen dem Menschen sichtbar und verständlich zu machen und verwenden hierzu geeignete visuelle Paradigmen, häufig verbunden mit entsprechenden Interaktionsmöglichkeiten.

Die vorliegende wissenschaftliche Arbeit ist in den Bereich der angewandten Computergrafik, speziell der interaktiven Visualisierung, einzuordnen. Die primären Ziele lagen dabei in der Übertragung des Begriffes *kontextsensitiv* auf den Bereich der Visualisierung zur Sicherstellung effizienter und kontextsensitiver Visualisierungsapplikationen sowie die Anwendung in aktuellen praktischen Aufgabenstellungen. Die Umsetzung einer kontextsensitiven Visualisierung gelingt im Rahmen dieser Arbeit durch die zukunftsweisende Kopplung von Visualisierungspipeline und Agententechnologie. Basierend auf der Identifikation zentraler Szenarien der kontextsensitiven Visualisierung wird eine agentenbasierte Visualisierungskontrolle durch intelligente Überwachung und Regelung der Visualisierungspipeline vorgestellt.

Nach einer Zusammenfassung der relevanten Grundlagen aus den Gebieten der Visualisierung und der Agententechnologie folgen eine theoretische Klassifizierung und ein Überblick über existierende Systeme und Anwendungen aus beiden Bereichen. Anschließend wird das im Rahmen dieser Arbeit erarbeitete Paradigma der kontextsensitiven Visualisierung vorgestellt und die praktische, komponentenbasierte Umsetzung erläutert.

Einen nicht unerheblichen Anteil der Arbeit machen drei innovative, auf der kontextsensitiven Visualisierung basierende Visualisierungsapplikationen aus, welche die Möglichkeiten und die Funktionsfähigkeit der entwickelten Architektur aufzeigen. Die Entwicklung einer plattformunabhängigen interaktiven Visualisierung beschäftigt sich insbesondere mit dem Auffinden der aktuell maximal möglichen Performance durch Abwägung der gegenläufigen Hauptparameter Qualität und Interaktivität und behandelt damit vor allem den System- und Interaktionskontext. Der Gedanke der plattformunabhängigen interaktiven Visualisierung wird anschließend auf mobile Informationssysteme ausgeweitet. Hier ist neben den Performanceaspekten vor allem die Art des Ausgabemediums, d. h. der Darstellungskontext, ein entscheidender Faktor. Die dritte Anwendung stellt eine agentenbasierte Applikation für die Bekleidungsindustrie in Form eines interaktiven Individual-Katalogs dar und behandelt insbesondere den Daten- und den Benutzerkontext.

Eine kurze Zusammenfassung sowie ein Ausblick auf geplante zukünftige Entwicklungen runden letztlich die Betrachtungen ab.

Inhaltsverzeichnis

1
Einleitung

1.1 Einführende Bemerkungen

Zu Beginn der 90er Jahre war die Ausführung komplexer Visualisierungsapplikationen ausschließlich auf Systemen mit extrem teurer Spezialhardware und geeignet angepasster Software möglich. Als Folge entstanden hauptsächlich auf Workstation-Basis viele große, monolithische Systeme und Insellösungen. Im Laufe der Zeit sind diese Systeme selbst für Spezialisten aufgrund der Größe und der funktionalen Abhängigkeiten kaum noch überschaubar geworden. Die Fehleranfälligkeit der Systeme stieg mit zunehmender Komplexität bei gleichzeitig abnehmender Wartbarkeit.

Die drastisch gesunkenen und immer noch weiter fallenden Kosten für leistungsfähige Prozessoren, Speicherchips, Massenspeicher und insbesondere hochperformante Grafikhardware verleihen aktuellen PCs für den Consumer-Bereich Rechen- und Grafikleistung, die selbst Workstations übertreffen, welche noch vor wenigen Jahren als State of the Art galten. Zudem hat der extreme Preisverfall zu einer rapiden Verbreitung von PCs auch in den privaten Haushalten geführt. Zeitgleich entwickelten sich das Internet und das World Wide Web als Informationsplattform geradezu explosionsartig. Diese Entwicklung ist weiter ungebrochen, sie hat sich im Gegenteil durch die Einführung von schnellen Internetzugängen (DSL, ein breitbandiger Internetzugang mit Übertragungsraten von bis zu 2 Mbit/s) und der starken Verbreitung von mobilen Geräten wie Multimedia-fähigen Handys und PDAs sogar noch verstärkt.

Die damit gebotene überwältigende Fülle an Informationen ist für den Menschen ohne geeignete Hilfsmittel nur schwer überschau- und begreifbar. Die Tatsache, dass der Mensch wesentlich besser mit visuellen Eindrücken als mit abstrakten Zahlen oder Fakten umgehen kann, hat daher maßgeblich zur Entwicklung des äußerst dynamischen Forschungs- und Anwendungsgebietes der Visualisierung beigetragen. Entsprechend wächst auch die Anzahl der verfügbaren Visualisierungsanwendungen stetig an. Beispielhaft seien hier die Forschungs- und Anwendungsbereiche der Medizinischen Visualisierung, der Web-basierten Visualisierung, der Visualisierung großer unstrukturierter Datenmengen, der Information Visualization und der Scientific Visualization genannt.

In der Regel sind die entwickelten Visualisierungssysteme und -applikationen jedoch stark auf spezielle Anwendungen ausgerichtet und zeigen daher vor allem Mängel bei der Flexibilität des Visualisierungsprozesses. Je nach Applikation oder Konfiguration bieten sie dem Benutzer entweder eine hohe Darstellungsqualität mit eingeschränkten Interaktionsmöglichkeiten oder eine Echtzeitvisualisierung verbunden mit einer verringerten Renderingqualität an. Zudem sind die Systeme oft nicht oder nur eingeschränkt auf verschiedenen Plattformen und Hardwarekonfigurationen lauffähig. Auch die benutzerindividuellen Anpassungsmöglichkeiten der Applikationen sind meistens auf die Einstellung rein optischer Features der Benutzeroberfläche oder den Zuschnitt bestimmter Programmversionen auf gewisse Benutzergruppen beschränkt.

Die beschriebene Entwicklung ist jedoch völlig gegenläufig zu der in der heutigen Informationsgesellschaft zunehmend stärker aufkommenden Forderung nach besonderer Berücksichtigung von Individualität und Personalisierung. Hierzu zählt nicht nur die individuelle Ausrichtung der Applikationen auf die jeweiligen Interessen und Fähigkeiten eines Benutzers (d. h. die Adaption an den aktuellen Benutzer- und Situationskontext), sondern auch das reibungslose Funktionieren auf einer Vielzahl von Plattformen bei gleichzeitig möglichst konsistenter Bedienbarkeit und identischem Informationsangebot (d. h. die Adaption an System-, Interaktions- und Darstellungskontext).

Einen Ausweg aus dieser Problematik bietet die im Rahmen dieser Arbeit vollzogene Übertragung des Begriffes *kontextsensitiv* auf den Bereich der Visualisierung zur Sicherstellung effizienter und kontextbezogener Visualisierungsapplikationen.

1.2 Inhalt

Die vorliegende wissenschaftliche Arbeit beschäftigt sich mit der innovativen Thematik der Verbindung der beiden Forschungsgebiete Visualisierung und Agententechnologie. Primäre Ziele dieser Arbeit waren der Entwurf eines zukunftsorientierten Ansatzes für die kontextsensitive Visualisierung und dessen Anwendung in innovativen, praktischen Aufgabenstellungen aus dem ingenieur- und naturwis-

senschaftlichen Bereich. Eine besondere Forderung bei der Entwicklung war die Verwendbarkeit der entstehenden Visualisierungslösungen auf beliebigen skalierbaren Hardwareplattformen unter optimaler Ausnutzung dort lokal vorhandener Ressourcen, angefangen von Handys und PDAs bis hin zu Grafik-Workstations.

Kapitel 2 (Grundlagen: Visualisierung, Agenten- und Komponentensysteme) behandelt die für die Arbeit relevanten Grundlagen aus den Forschungsgebieten Visualisierung und Agentensysteme. Insbesondere wird bereits hier auf eine komponentenorientierte Sichtweise geachtet (Bausteine der Visualisierungspipeline, Multi-Agenten-Systeme und Kommunikationsmethoden) sowie eine kurze Einführung in die bei der Umsetzung angewandte Komponententechnologie gegeben.

Ab Kapitel 3 (Kontextsensitive Visualisierung) folgen die Ausführungen zu den in dieser Arbeit vorgestellten neuen Resultaten. Nach einer Untersuchung und Bewertung existierender Visualisierungsparadigmen und Anwendungen von Agenten in Softwaresystemen wird zunächst der Begriff der kontextsensitiven Visualisierung eingeführt und definiert. Anschließend werden konkrete Kontext-Szenarien in der Visualisierung identifiziert und erläutert. Neben der Erläuterung der Struktur einer agentenüberwachten- und gesteuerten Visualisierungspipeline enthält dieses Kapitel auch Ausführungen zur praktischen, komponentenbasierten Umsetzung.

Kapitel 4 (MacVis – System- und Darstellungskontext) beschreibt zunächst unabhängig von der Anwendung die Realisierung einer agentenbasierten Visualisierungskontrolle durch ein geeignetes Multi-Agenten-System. Die beschriebenen Techniken werden dann zunächst zur Visualisierung großer Datenmengen, speziell in der medizinischen Informatik (Computer-Tomografie), an einem praktischen Beispiel verifiziert. Eine weitere Evaluierung erfährt der vorgestellte Ansatz durch die Konzeption und Entwicklung eines skalierbaren Informationsvisualisierungssystems auf mobilen Endgeräten, dargestellt am Beispiel der effizienten Informationsvermittlung und betrieblichen Leistungssteigerung im Bereich von Abwasseranlagen.

In Kapitel 5 (Virtual Try-On – Daten- und Benutzerkontext) wird eine agentenbasierte Visualisierungsapplikation für die Bekleidungsindustrie vorgestellt. Das zentrale Ziel der beschriebenen Virtual Try-On Anwendung liegt in der Konzeption und Umsetzung eines interaktiven Individual-Katalogs, d. h. einem Online-Katalog, in dem der Kunde sich selbst als 3D-Modell in der gewählten Bekleidung betrachten kann und welcher trotzdem ein hohes Maß an Interaktivität durch entsprechend schnelle Berechnungs- und Darstellungsmethoden erlaubt. Die gängigen Techniken zur Bekleidungssimulation sind für diese Aufgabenstellung nicht anwendbar, da mit diesen eine im Rahmen der Kundenakzeptanz liegende Reaktionszeit nicht erreicht werden kann. Eine Lösung liefert die in dieser Arbeit vorgestellte regelbasierte Morphingtechnik.

Ein kurzes Resümee sowie ein Ausblick auf geplante zukünftige Entwicklungen runden letztlich mit Kapitel 6 (Zusammenfassung und Ausblick) die Betrachtungen ab.

1.3 Allgemeines

Diese Arbeit entstand innerhalb der letzten Jahre im Rahmen meiner Tätigkeit als wissenschaftlicher Mitarbeiter im von Prof. Dr. Hans Hagen geleiteten Forschungsbereich *Intelligente Visualisierung und Simulation* an der Deutsches Forschungszentrum für Künstliche Intelligenz (DFKI) GmbH. Zusätzlich zu dem hier vorliegenden Teil der schriftlichen Ausarbeitung besteht sie aus einem nicht unbeträchtlichen praktischen Anteil, der die Umsetzung der beschriebenen Inhalte in Programmcode umfasst. Die entstandenen Quelltexte sind indes sehr komplex und umfangreich, so dass auf einen Abdruck – auch auszugsweise – bewusst verzichtet wurde. Stattdessen belegen neben Erklärungen der strukturellen Funktionsweise der entwickelten Komponenten viele beispielhaft berechnete Bilder die Möglichkeiten der umgesetzten praktischen Systeme.

Mittlerweile existieren zu vielen englischen Fachausdrücken und Eigennamen aus dem Gebiet der Informatik, speziell dem hier betroffenen Bereich der praktischen Computergrafik und Visualisierung, deutschsprachige Pendants. Wegen der fehlenden Wiedererkennung aufgrund mangelnder Verbreitung wird in dieser Arbeit bei gebräuchlichen englischen Schlagworten und Fachausdrücken absichtlich auf eine Übersetzung verzichtet. Von einer besonderen Kennzeichnung, z. B. durch Kursivschrift oder Anführungszeichen, wird ebenfalls abgesehen.

2

Grundlagen: Visualisierung, Agenten- und Komponentensysteme

Das folgende Kapitel liefert die Grundlagen für die weiteren Abschnitte dieser Arbeit. Zwei Begriffe stehen dabei im Mittelpunkt: die Scientific Visualization und die Agentensysteme. Bemerkungen zur modernen Anwendungsentwicklung durch den Einsatz von Komponententechnologie und Visual Prototyping runden das Kapitel ab.

2.1 Visualisierung

In ihrer einfachsten Form ist die Visualisierung nichts anderes als die visuelle Repräsentation von Daten. Entsprechende Definitionen finden sich in vielen Lexika, als Beispiel soll hier die im Duden gegebene Begriffsbildung zitiert werden: „visualisieren <lat.> (optisch darstellen)“ [20]. Nach einigen Bemerkungen zu den historischen Hintergründen und zur Scientific Visualization allgemein, werden zwei praktische, anwendungsorientierte Sichten auf den Visualisierungsprozess beschrieben: der Visualization Cycle und die Visualization Pipeline.

2.1.1 Historisches

Bereits weit vor der Erfindung des Computers und der heute meistens damit verbundenen Begriffsbildung der Visualisierung lassen sich erste Ansätze zur visuellen Darstellung von Daten finden.

Ein Beispiel hierfür stellen die Untersuchungen von Dr. John Snow dar, die dieser während des Ausbruchs der Cholera in London in den Jahren 1853-54 durchführte. Aufgrund der ihm bekannten Daten konnte er zwar feststellen, dass eine starke Häufung der Krankheitsfälle im Stadtteil Soho auftrat, sein Ziel lag jedoch mehr in der Ursachenfindung. Hierzu zeichnete er die Lage der Häuser von 500 Opfern, die während der ersten 10 Tage des Septembers 1854 gestorben waren, auf einer Karte des Gebiets ein. Diese einfache visuelle Darstellung der Daten zeigte eine Häufung der Krankheitsfälle rund um eine einzelne Wasserpumpe. Anschließende Untersuchungen dieser Pumpe ergaben, dass das Wasser durch eine undichte Jauchegrube verunreinigt war.

2.1.2 Scientific Visualization

Aus diesem exemplarischen Beispiel lässt sich gut erkennen, dass man bereits durch eine einfache Transformation von Daten in eine einfache grafische Darstellung ansonsten nicht erfahrbare, tief greifende und weiterführende Einblicke in die zugehörige Problemstellung erhalten kann. Entsprechend liegt das zentrale Ziel der Scientific Visualization in der Unterstützung von Wissenschaftlern beim Sichten, Auswerten und Verstehen ihrer Daten. Die Daten entstehen dabei häufig bei Experimenten und numerischen Simulationen und sind bei einer direkten Untersuchung aufgrund ihrer Größe oder ihrer Komplexität selbst von Experten nicht oder nur schwer direkt zu interpretieren. Die Problematik verschärft sich weiter, wenn die Daten sich zusätzlich noch über die Zeit ändern.

Durch ständig wachsende Rechenleistungen moderner Computer und gleichzeitig fallende Preisen erhält die Computertechnologie Einzug in immer mehr Bereiche der Wissenschaft. Viele Vorgänge, die früher an kosten- und zeitintensiven Prototypen durchgeführt wurden, lassen sich heute mittels rechnergestützter Simulationen schneller und günstiger durchführen. Durch die Vielzahl neuer Anwendungsfelder und die aufgrund hoher Rechenleistungen immer größer werdenden Datenmengen steigen jedoch gleichzeitig auch die Anforderungen an die Algorithmen zur computergrafischen Darstellung der Daten. Neben neuen oder verbesserten Methoden und Techniken zum Filtern und Clustern der eigentlichen Daten stellen sich hier vor allem stark erhöhte Anforderungen an die Interaktionsmöglichkeiten bei der Visualisierung.

Für die Visualisierung von Daten existieren verschiedenste Ansätze. Eine einfache und kostengünstige Darstellung bieten die 2D-Graphen, die sich gut für die Analyse kleinerer Datenmengen eignen. Komplexere Datensätze erfordern dagegen in der Regel eine dreidimensionale Visualisierung. Hier lassen sich statische 3D-Darstellungen, Animationen, stereografische sowie interaktive Visualisierungen unterscheiden. Dabei ist die Form der Visualisierung nicht nur abhängig von Größe und Art der zugrunde liegenden Daten, sondern vor allem auch durch die

Art und Situation der Anwender bzw. Betrachter und den angestrebten Verwendungszweck beeinflusst.

Der eigentliche Visualisierungsprozess lässt sich in sechs Schritte unterteilen:

- Erzeugung der Daten, z. B. durch numerische Simulation
- Bestimmung der Art der Visualisierung basierend auf Art und Größe der Daten sowie den Anforderungen der Betrachter
- Filterung der Daten, z. B. durch Reduzierung der Datenmenge
- Konvertierung der Simulationsdaten in Geometriedaten
- Rendern der Geometrie
- Überprüfung des Ergebnisses der Visualisierung, z. B. in Bezug auf Anspruch und Exaktheit

Visualization Cycle und Visualization Pipeline geben eine mehr anwendungsorientierte Sicht auf diesen Prozess.

2.1.3 Visualization Cycle

Das von Upson et al. [120] entwickelte Scientific Visualization Modell beruht auf der Ähnlichkeit der Abläufe zur Erzeugung einer visuellen Datenrepräsentation und der Generierung der eigentlichen Daten.

Beim Prozess einer numerischen Simulation (siehe Abbildung 2-1) müssen die zugrunde liegenden physikalischen Bedingungen und Gleichungen zunächst in ein auf einem Computer ausführbares Programm überführt werden. Die so erzeugten berechenbaren Modelle müssen dann noch näher spezifiziert werden (z. B. durch Angabe der Start- und Randbedingungen). Im nächsten Schritt wird eine Lösung des gegebenen Problems berechnet und nachfolgend analysiert.

Der weitere Ablauf hängt nun vom Ergebnis der Analyse ab. Sind die erzeugten Daten beispielsweise zu ungenau oder sogar fehlerhaft, so müssen Änderungen im Programm oder bei der Spezifikation gemacht werden, d. h. der Computational Cycle wird von neuem durchlaufen. Sind bei der Analyse keine Probleme erkannt worden, so werden die Ergebnisse abschließend vom Wissenschaftler zusammengefasst.

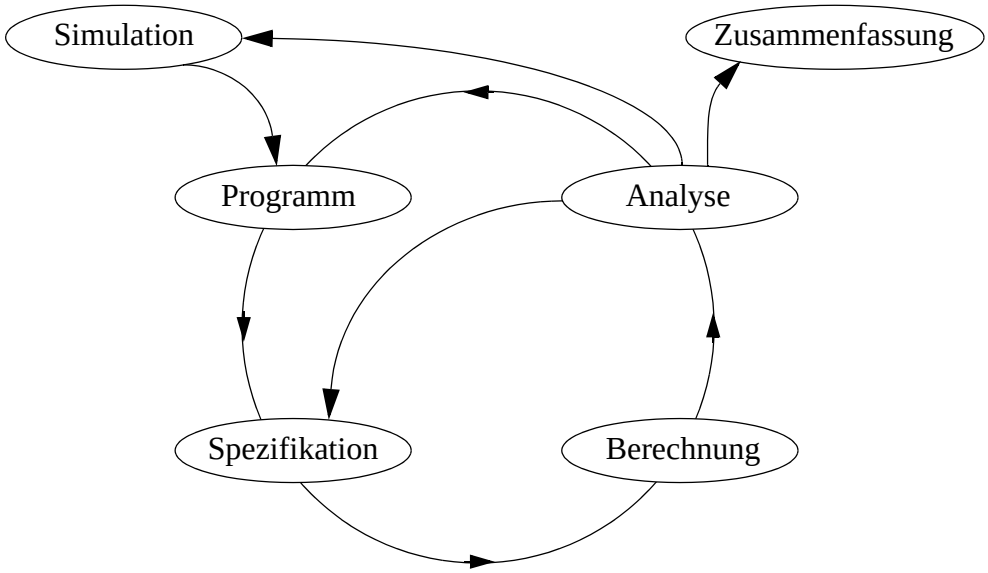

Abbildung 2-1: Numerischer Simulationsprozess (Computational Cycle).

Die Visualisierung der Daten stellt ein entscheidendes Hilfsmittel im Analyseschritt dar. Dieser Schritt kann nun wiederum in mehrere, einen Kreislauf bildende Abschnitte unterteilt werden – den Analysis Cycle (siehe Abbildung 2-2).

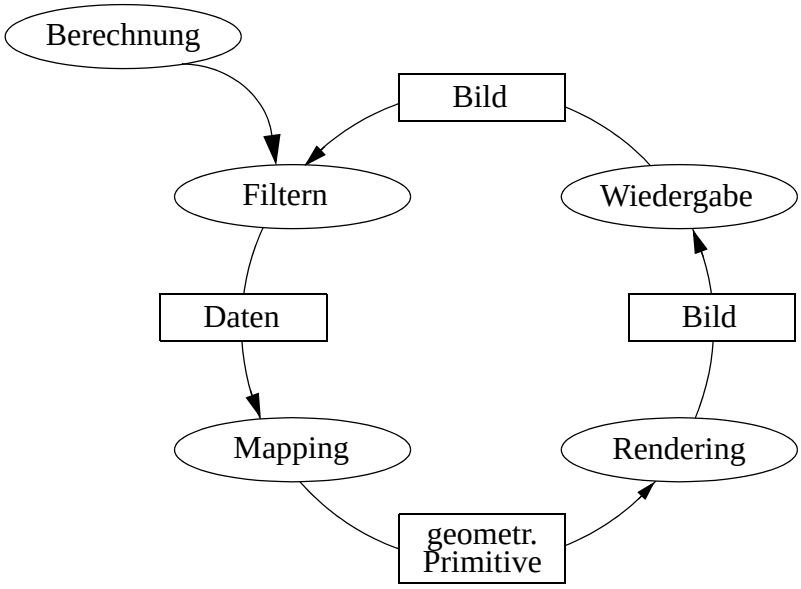

Abbildung 2-2: Analysis Cycle.

Zunächst werden die während der Berechnung erzeugten Daten einem Filterprozess unterworfen, beispielsweise zur Datenreduktion. Die resultierenden Daten werden dann auf entsprechende geometrische Primitive abgebildet und abschlie-

ßend mittels Rendering die zugehörige Visualisierung erzeugt. Im letzten Schritt (Wiedergabe) des Kreislaufs kann das erzeugte Bild nun mit den erwarteten oder bereits berechneten Ergebnissen verglichen werden. Ist das Resultat nicht zufrieden stellend, so startet der Analysis Cycle mit entsprechenden Anpassungen von neuem.

Das Zusammenspiel von Computational und Analysis Cycle ist als Visualization Cycle bekannt.

2.1.4 Visualization Pipeline

Ein weiteres, häufig verwendetes Visualisierungsmodell ist die so genannte Visualization Pipeline, die erstmals von Haber und McNabb [44] vorgestellt wurde. Diese teilt den Visualisierungsprozess in drei Schritte auf: Data Enrichment and Enhancement, Visualization Mapping und Rendering. Im ersten Schritt werden die Simulationsdaten z. B. durch Filtern in ein für die Visualisierung geeignetes Format gebracht. Anschließend werden diese Daten dann in ein so genanntes abstraktes Visualisierungs-Objekt (AVO) umgesetzt, das neben den eigentlichen Daten noch zusätzliche Informationen beinhalten kann (z. B. Farbe, Zeitstempel usw.). Im abschließenden Schritt wird das AVO gerendert, d. h. das AVO wird mittels Techniken der Computergrafik in ein Bild umgesetzt.

Abbildung 2-3 zeigt eine gegenüber dem Modell von Haber und McNabb leicht erweiterte Fassung der Visualization Pipeline bestehend aus den Stufen Simulation (I), Visualization (II) – hier gehen die ersten beiden Stufen von Haber und McNabb ein – und Rendering (III). Die Simulationsstufe erzeugt aus gegebenen Messwerten die Simulationsdaten, die in der Visualization-Stufe in Visualisierungsdaten transformiert werden. Die Rendering-Stufe erzeugt schließlich die zugehörigen Bilddaten. Es ist anzumerken, dass die aus den Stufen (II) und (III) gebildete Subpipeline auch unter dem Namen Rendering-Pipeline bekannt ist. Im Folgenden werden die einzelnen Stufen näher betrachtet.

In die Stufe (I) gehen vor allem natur- oder ingenieurwissenschaftliche Grundlagen ein. Basierend auf Messdaten oder Erfahrungen wird hier ein (mathematisches) Modell der gegebenen Problemstellung aufgestellt. Ausgabe der ersten Pipelinestufe sind die von der numerischen Simulation hieraus generierten Simulationsdaten, die ihrerseits wieder die Eingabe der Stufe (II) sind. Da die Simulationsdaten aufgrund ihrer Größe und Multidimensionalität in der Regel nicht zur direkten Visualisierung geeignet sind, werden hier zunächst in einem Filter-Prozess die relevanten Informationen extrahiert und – falls nötig – die Menge der Daten durch geeignete Reduktionsverfahren verkleinert. Im anschließenden Mapping-Prozess wird nun ein auf den Daten und der Problemstellung basierendes Visualisierungsmodell berechnet. Die aus diesem Modell hervorgehenden Visuali-

sierungsdaten sind lediglich Geometriebeschreibungen wie zum Beispiel geometrische Primitive, die dann in der Pipelinestufe (III) zunächst diskretisiert werden. Die diskretisierten Daten, zum Beispiel in Form von Dreiecken, werden dann basierend auf der Sichtdefinition transformiert. Im nächsten Schritt, der Projektion, entsteht dann durch Rasterisierung das eigentliche (Pixel-)Bild, d. h. die Bilddaten.

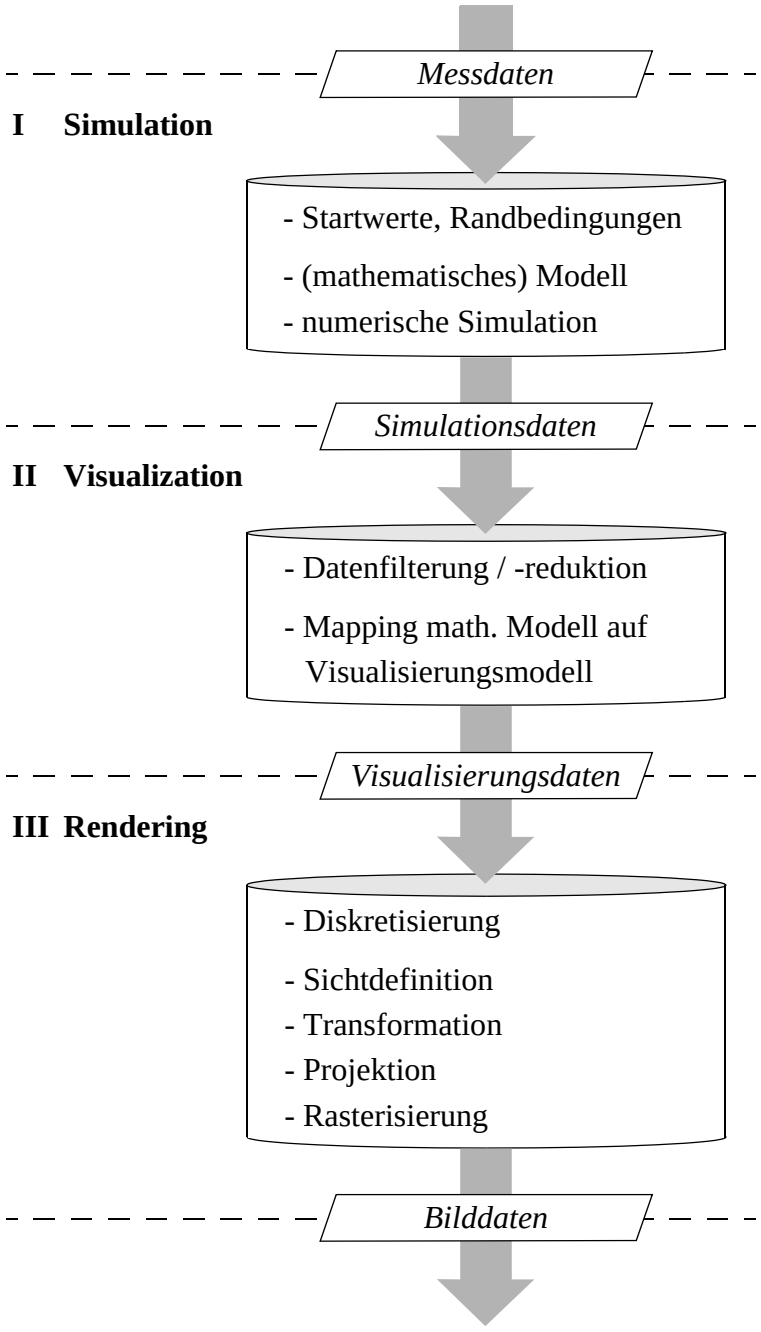

Abbildung 2-3: Scientific Visualization Pipeline.

Es ist offensichtlich, dass jede Stufe vom Ergebnis der vorhergehenden abhängig ist. Die für den Visualisierungsprozess benötigte Zeit ergibt sich damit aus der Summe der Ausführungszeiten der einzelnen Stufen. Prinzipiell kann der Benutzer bei jeder Stufe der Pipeline interaktiv eingreifen. In der Regel findet die eigentliche Interaktion jedoch hauptsächlich in Stufe (III) statt; die Schritte Sichtdefinition, Transformation, Projektion und Rasterisierung werden entsprechend in einer Schleife ausgeführt.

2.2 Agenten-Systeme

(Multi-)Agenten-Systeme (MAS) tragen häufig auch die Bezeichnung „Verteilte Künstliche Intelligenz (VKI)“. Schon aus dieser Namensgebung lassen sich zwei tragende Säulen der Multi-Agenten-Systeme erkennen. Zum einen ist dies das Gebiet der Künstlichen Intelligenz (KI) mit zentralen Forschungsaspekten wie Wissensrepräsentation, Lernen oder Planen. Zum anderen basiert die VKI maßgeblich auf den Verteilten Systemen, auf denen beispielsweise die Kommunikationskonzepte der MAS beruhen.

Zentrale Grundlage der MAS bildet der (Software-)Agent selbst, dessen Konzept stark am Vorbild des Menschen orientiert ist. Der Anwender gibt einem Agenten eine bestimmte Aufgabe, die dieser dann selbständig lösen soll. Hierzu muss der Agent zum Beispiel die zur Aufgabenlösung nötigen Informationen suchen, sammeln und auswerten und das Ergebnis schließlich dem Auftraggeber präsentieren. Die Hauptziele beim Einsatz von Agenten sind dabei die Erreichung von größerer Effizienz, schnelleren Antworten und besseren Lösungen. Die Möglichkeit der Kommunikation und Kooperation der einzelnen Agenten in MAS unterstützt zudem die Ausnutzung von Synergieeffekten.

In den folgenden Ausführungen werden zunächst die nötigen Begriffe im Kontext der Agenten-Systeme definiert und auf Eigenschaften und Kommunikationsmethoden von Agenten eingegangen.

2.2.1 Agenten: Definitionen und Eigenschaften

Im allgemeinen Gebrauch bezeichnet man mit einem Agenten eine Person, die im Auftrag oder Interesse eines anderen tätig ist [80]. Beispiele für Agenten sind Handelsvertreter, Versicherungsvertreter oder Diplomaten und Spione als „Vertreter“ eines Staates. Das Wort Agent geht zurück auf den lateinischen Begriff „agere“, der mit „handeln“ oder „tätig sein“ übersetzt wird.

Die exakte Angabe einer Definition für einen Softwareagenten gestaltet sich dagegen schwierig. In der Literatur gibt es bis heute keine allgemein anerkannte, ein-

deutige Begriffsbildung – die verschiedenen Definitionen und Klassifizierungen bauen nur selten aufeinander auf bzw. schließen sich sogar gegenseitig aus. Hinzu kommt, dass Agenten in den verschiedensten Anwendungsgebieten zum Einsatz kommen und hier jeweils eigene oder angepasste Begriffsbildungen verwendet werden. Auch der aktuelle Trend in der Softwareindustrie, aufgrund der Popularität des Agentenbegriffes jede neu entwickelte Softwarekomponente aus Marketinggründen als Agenten zu bezeichnen, trägt weiter zur Verwässerung des Begriffes bei.

Formal lässt sich ein Agent wie folgt definieren (siehe [18]):

> *Ein Agent A ist ein Tripel (Dat, Akt, Sit) mit einer internen Datenbasis Dat (einer Menge von unstrukturierten Daten zur Wissensrepräsentation), einer Menge von Aktionen Akt und einer Menge von Situationen Sit.*
> *A ist also eine Funktion f_A: Sit -> Akt, d. h. ein Agent bildet eine bestimmte Situation auf eine zugehörige Aktion ab. Intern ist hierzu natürlich auch die Abhängigkeit zur Datenbasis Dat gegeben, so dass hier A: Sit x Dat -> Akt gilt.*

Jennings und Woolridge [64] definieren in ihrer, in der Fachliteratur auch als „Weak Notion of Agency“ bezeichneten Auffassung den Agentenbegriff mit Hilfe von Eigenschaften, die ein System haben muss, um als Agent bezeichnet werden zu können. Die dort gemachte Unterscheidung zwischen *agent* und *intelligent agent* soll in dieser Arbeit jedoch keine Anwendung finden, ein Agent wird im vorliegenden Kontext immer mit dem Schlüsselwort *Intelligenz* verknüpft gesehen.

> *Ein Agent ist ein Computersystem, das flexible, autonome Aktionen zur Erreichung seiner Zielstellung ausführen kann. Flexibel bedeutet in diesem Zusammenhang, dass das System immer ansprechbar (responsive), voraushandelnd (proactive) und sozial (social) ist.*

Die hier hervorgehobenen Eigenschaften eines Agenten sollen nun näher spezifiziert werden:

- *Autonomie*: Ein Agent ist autonom, wenn er eigenständig seine Ziele verfolgt, d. h. ohne das direkte Einschreiten des Anwenders. In der Literatur finden sich auch für die Definition der Autonomie verschiedene Auffassungen. Oftmals dürfen autonome Agenten nicht von anderen Agenten gesteuert werden. Im Rahmen dieser Arbeit wird jedoch in einer solchen Steuerung keine Verletzung der Autonomie gesehen, solange der gesteuerte Agent einen gewissen Grad an Eigenständigkeit – wenn auch eingeschränkt – behält.

- *Ansprechbarkeit, Reaktivität*: Ein Agent ist reaktiv, wenn er auf Veränderungen in seiner Umwelt reagieren kann. Eine solche Veränderung kann zum Beispiel durch eine Interaktion des Anwenders mit dem System ausgelöst werden.
- *Pro-Aktivität*: Agenten sind voraushandelnd, wenn sie nicht nur auf Veränderungen reagieren können, sondern auch selbst eine Handlungsinitiative ergreifen können. Eine andere, oft in der Literatur verwendete Bezeichnung für dieses Verhalten ist zielorientiert (goal-oriented).
- *Soziales Verhalten*: Agenten sind sozial, wenn sie mit anderen Agenten kommunizieren und diese bei der Ausführung von Aktionen nicht behindern. Daher wird häufig synonym zur Bezeichnung „sozial“ auch der Begriff „kommunikativ“ verwendet.

Vergleicht man dies mit anderen Definitionen in der Fachliteratur, so zeigt sich als Gemeinsamkeit, dass Agenten fast immer zumindest zu einem gewissen Grad als autonom angesehen werden [37]. Die „Weak Notion of Agency“ erweitert die Autonomie durch die Hinzunahme der Flexibilität und kann durch die Hinzunahme weiterer Eigenschaften verschärft werden; man bezeichnet dann die Begriffsbildungen entsprechend als „Stronger Notions“. Nachfolgend sollen einige dieser Eigenschaften kurz vorgestellt werden:

- *Adaption*: Adaption bezeichnet die Anpassungsfähigkeit eines Agenten, d. h. inwieweit dieser aus den von ihm gemachten Erfahrungen dazulernen und sein Handeln entsprechend anpassen oder verbessern kann. Adaptive Agenten werden daher auch als lernend bezeichnet.
- *Deliberativ*: Deliberative oder zielgerichtete Agenten besitzen Wissen über ihre Umgebung und können aufgrund ihres Planungsvermögens Aktionen ausführen in der Hoffnung, ein bestimmtes Ziel zu erreichen.
- *Kommunikation*: Besitzt ein Agent diese Eigenschaft, so kann er mit anderen Agenten oder auch mit dem Anwender Informationen und Wissen austauschen.
- *Mobilität*: Ein Agent ist mobil, wenn er nicht fest an einen bestimmten Computer gebunden ist, sondern selbständig von Rechner zu Rechner wandern kann, sofern dies zur Lösung seiner Aufgaben nötig ist.
- *Rationalität*: Ein Agent wird als rational bezeichnet, wenn er versucht, seine Ziele möglichst optimal und vollständig zu erreichen.
- *Reflektivität*: Ein Agent handelt reflektiv, wenn er bei seinen Aktionen hauptsächlich auf seine interne Datenbasis vertraut. Im Gegensatz zur Reaktivität kann er dadurch schlechter auf externe Situationen reagieren.

- *Unabhängigkeit*: Je weiter die an einen Agenten gestellten Aufgaben gefasst sind, desto freier (d. h. desto unabhängiger) ist dieser bei den zur Ergebnisfindung zu treffenden Entscheidungen.

Agenten werden heute in den unterschiedlichsten Anwendungsbereichen und Ausprägungen eingesetzt. Da die obigen Definitionen relativ allgemein gehalten sind, erscheint es sinnvoll, die Agenten in unterschiedliche Klassen einzuteilen. Neben der Unterteilung der einzelnen Agenten nach ihren Eigenschaften finden sich in der Literatur vielfältige, davon abweichende Versuche zur Klassifizierung der verschiedenen Agententypen.

Franklin und Graesser [37] führen hierzu eine an biologische Modelle angelehnte natürliche Taxonomie für Agenten ein (siehe Abbildung 2-4). Die „Abstammung“ der im Kontext dieser Arbeit wichtigen, aufgabenspezifischen Agenten ist in der Abbildung hervorgehoben.

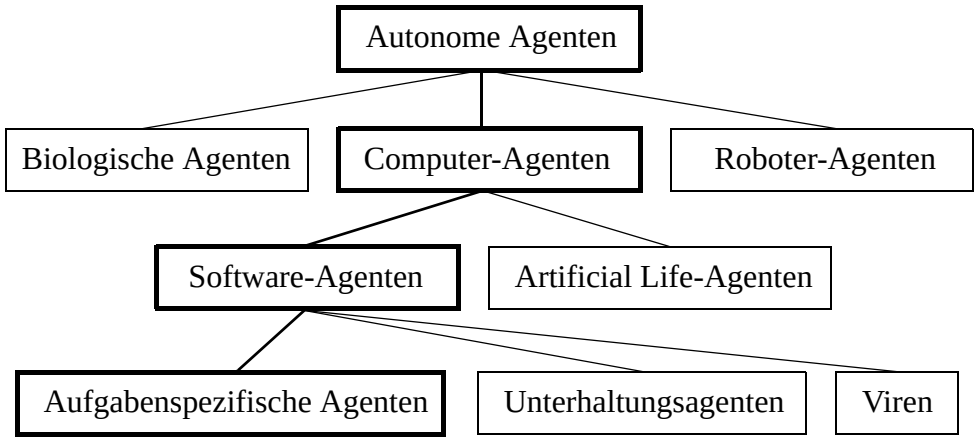

Abbildung 2-4: Natürliche Taxonomie nach Franklin und Graesser.

Weitere Möglichkeiten der Klassifizierung stellt Nwana in [88] vor. Nwanas Definition eines Agenten entspricht dabei nicht der „Weak Notion of Agency“, es werden nur drei zentrale Attribute verwendet, die ein Agent haben sollte: autonom, kooperativ und lernend. Dabei ist jedoch keines der Attribute zwingend nötig. In der in Abbildung 2-5 angegebenen Typologie können somit Agenten beispielsweise auch nicht-autonom sein.

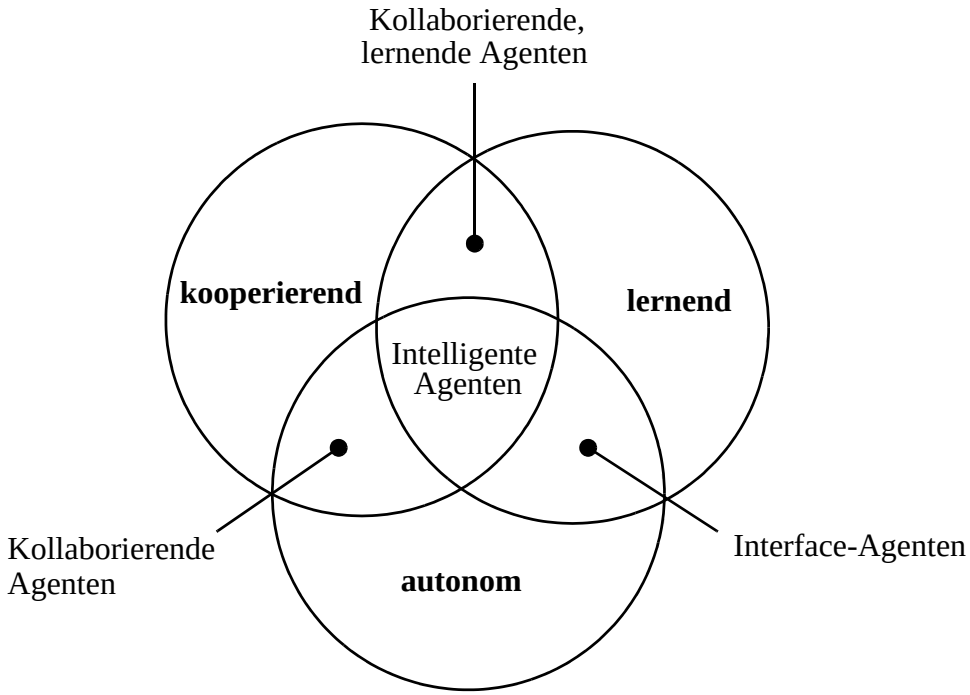

Abbildung 2-5: Agenten-Typologie nach Nwana.

Nwana gibt in [88] noch eine weitere Möglichkeit zur Klassifizierung an; hier spielen vor allem die verschiedenen Eigenschaften der Agenten eine Rolle. Die Agenten-Typologie unterteilt sich hiernach in:

- Kollaborierende Agenten
- Interface-Agenten
- Mobile Agenten
- Informations- / Internet-Agenten
- Reaktive Agenten
- Hybride Agenten
- Intelligente Agenten

Wirken Agenten aus verschiedenen Kategorien in einer Applikation zusammen, so bezeichnet Nwana dies als ein heterogenes Agentensystem. Eine genaue Beschreibung der Funktionsweise und Ziele der einzelnen Agententypen dieser Klassifizierung findet sich ebenfalls in [88].

Auch für die Modellierung von Agenten gibt es viele verschiedene Ansätze, die im Folgenden kurz zusammengefasst werden.

Die einfachste Möglichkeit, einen Agenten zu modellieren, besteht in der Umsetzung als *Situations-Aktions-Paare*. Hierzu werden Situationen und dazugehörende Aktionen in Form einer Liste fest implementiert. Es ist offensichtlich, dass ein solcher Agent nur auf eine endliche Menge von Situationen reagieren kann und keine interne Datenbasis benötigt. Je nach Komplexität der Aufgaben des Agenten und der dadurch möglichen Situationen kann die Situations-Aktions-Liste sehr groß werden. Ein erweiterter Ansatz (siehe [19]) beschränkt daher die Menge der Paare auf einige wenige prototypische Situationen. Tritt nun eine so genannte erweiterte Situation auf, d. h. eine nicht in der Liste vorkommende Voraussetzung, so wählt der Agent die ähnlichste Situation und führt die zugehörige Aktion aus. Die Auswahl der ähnlichsten Situation kann hier beispielsweise über die Definition eines Entfernungsmaßes und die Bestimmung des nächsten Nachbarn (nearest neighbor) erfolgen.

Eine verbreitete und schon frühzeitig eingeführte Form der Agentenmodellierung ist die *regelbasierte Modellierung*. Der Agent besitzt hier als interne Datenbasis eine Menge von (Zustands-)Variablen und Regeln. Die Regeln besitzen die Form

if <cond> then <act>

mit der Bedingung *cond* (zusammengesetzt aus einer Situation und dem aktuellen internen Zustand des Agenten) und einer Aktion *act*. Der Agent muss also aufgrund einer gegebenen Bedingung eine anwendbare Regel auswählen und diese dann anwenden. Bei einer regelbasierter Modellierung ist das Lernen neuer Regeln relativ einfach zu bewerkstelligen, so dass sich diese Modellierungsart in letzter Zeit wieder größerer Beliebtheit erfreut.

Grundlage der *zustandsorientierten Modellierung* bilden die Automatentheorie und die Petri-Netze. Ein Agent geht abhängig vom aktuellen Zustand und der aktuellen Situation in einen neuen Zustand über und durchläuft somit eine Reihe von Zuständen.

Bei der *objektorientierten Modellierung* eines Agenten besitzt dieser analog zur objektorientierten Programmierung genau zwei Operatoren: „Empfangen“ beschreibt die Menge der Situationen und „Senden“ die Menge der Aktionen. Durch die Kapselung der Funktionsweise des Agenten eignet sich diese Art der Modellierung sehr gut zur Entwicklung von Multi-Agenten-Systemen. Für einzelne Agenten, die nicht in einem Multi-Agenten-System eingesetzt werden sollen, ist dieser Ansatz jedoch wenig praktikabel.

2.2.2 Multi-Agenten-Systeme

Unter einem Multi-Agenten-System (MAS) versteht man die Zusammenschaltung und das daraus resultierende Zusammenwirken (Kooperation) mehrerer einzelner

Agenten. Um einen Agenten in einem MAS verwenden zu können, muss man die formale Definition aus Abschnitt 2.2.1 entsprechend erweitern (siehe auch [18]):

> *Ein Agent in einem MAS ist analog zur Definition eines einzelnen Agenten ein Tripel (Dat, Akt, Sit). Die interne Datenbasis Dat besteht aus der Menge der individuellen Daten des Agenten, der Menge der sicheren Informationen über andere Agenten und der Menge der Annahmen über andere Agenten. Die Aktionen unterteilen sich in die Menge der Eigenaktionen und die Menge der Kommunikations- und Kooperationsmethoden. Die Menge der Situationen wird zum einen durch die Umwelt, zum anderen durch andere Agenten des MAS beeinflusst.*

Mit Hilfe dieser Definition lässt sich nun auch das MAS formal beschreiben:

> *Ein Multi-Agenten-System ist ein 5-Tupel (Sit, A, Akt_M, a, L) bestehend aus der Menge der Situationen Sit, der Menge der zum MAS gehörenden Agenten A, der Menge der im MAS möglichen Aktionen Akt_M, einer Funktion a von Akt_M nach A und der Aktionssprache L.*
> *Die Funktion a bildet dabei die Menge der möglichen Aktionen auf die Menge der im MAS vorhandenen Agenten ab, d. h. sie ordnet jeder Aktion einen Agenten zu, der diese Aktion ausführt.*

Hat man nun die einzelnen Agenten und die Kommunikationssprache für ein solches Multi-Agenten-System entwickelt, stellt sich die Frage nach der Organisation der Zusammenarbeit. Hier bieten sich zwei völlig unterschiedlich Ansätze an: zum einen die direkte Kommunikation zwischen den Agenten und zum anderen die Unterstützung der Koordination durch eine spezielle Komponente.

Der Vorteil der direkten Kommunikation ist, dass die Agenten nicht auf das Vorhandensein, die Fähigkeiten und Einstellungen anderer Programme angewiesen sind. Nachteilig sind jedoch die hierdurch verursachten Kosten, sobald eine größere Zahl von Agenten involviert ist. In diesem Fall müssen nämlich Nachrichten via Broadcast zu allen anderen Agenten verschickt werden und jeder Agent muss zudem für jede Nachricht prüfen, ob diese für ihn von Interesse ist. Ein weiterer Nachteil ist die aufwendige Programmierung solcher Agenten, da jeder Agent alle Routinen für die Verhandlung mit anderen Agenten beinhalten muss.

Die Alternative zur direkten Kommunikation stellt die Organisation von Agenten in so genannten Verbundsystemen (federated systems) dar. In diesem Fall kommunizieren die Agenten nie direkt miteinander, sondern verwenden hierfür eine spezielle Systemkomponente des jeweiligen Verbunds, den so genannten *facilitator*. Die Kommunikation von Agenten unterschiedlicher Verbünde erfolgt dabei über

die beiden entsprechenden Facilitators. Abbildung 2-6 zeigt die Struktur eines solchen Systems am Beispiel drei kooperierender Verbünde [40].

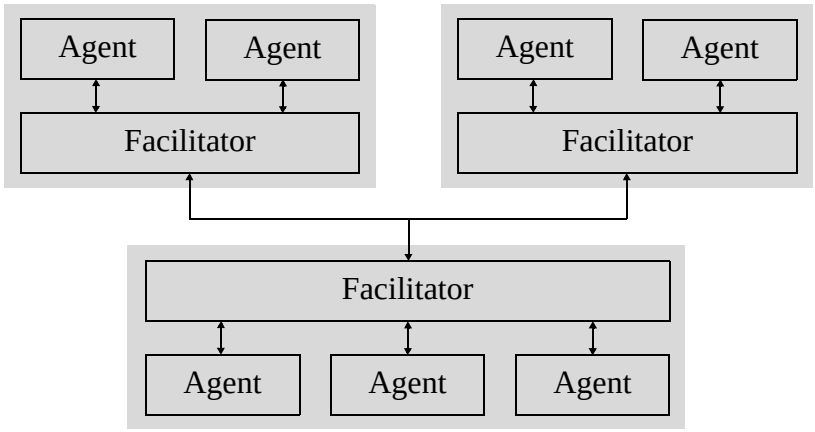

Abbildung 2-6: Realisierung von MAS als Verbundsysteme.

Ein Vorteil der Anwendung eines Multi-Agenten-Systems im Vergleich zur Verwendung eines einzelnen Agenten besteht in der Tatsache, dass kleine interagierende Agenten einfacher zu handhaben sind und zum Beispiel fehlerhafte Bausteine wesentlich leichter identifiziert und ausgetauscht werden können. Zudem liefert dieses Modell die Grundlage für verteiltes / paralleles Rechnen.

2.2.3 Agentenplattformen

Um die Zusammenarbeit verschiedener Agentensysteme sowie die Mobilität von Agenten zu ermöglichen, ist es erforderlich, hierfür allgemeine Spezifikationen und eine standardisierte Infrastruktur zur Verfügung zu stellen. Daher wurde 1996 die Foundation of Physical Intelligent Agents (FIPA) zur Erstellung von Softwarestandards für heterogene und interagierende Agenten sowie agentenbasierte Systeme gegründet. Unter den über 60 Mitgliedern der FIPA befinden sich sowohl Universitäten als auch namhafte Firmen wie z. B. HP, IBM, Intel, Siemens und Sun.

In der FIPA Agenten-Management-Spezifikation ([33], [34]) stellt die FIPA ein Framework für eine einheitliche Agentenplattform vor (siehe Abbildung 2-7). Das Agenten-Management Referenzmodell besteht hier aus den folgenden, logischen Komponenten:

- Die *Agentenplattform* (AP) stellt die gesamte physikalische Infrastruktur für die Agenten zur Verfügung. Zur AP gehören der Rechner, das Betriebssystem, jegliche die Agenten unterstützende Software, die FIPA Managementkomponenten (DF, AMS, ACC), der IPMT und die Agenten selbst.

- Der *Agent* ist das fundamentale Element einer Agentenplattform und stellt einen oder mehrere Dienste über ein einheitliches, integriertes Ausführungsmodell zur Verfügung. Dies kann zum Beispiel den Zugriff auf externe Software, den Benutzer und externe Kommunikationseinrichtungen beinhalten. Der Agent muss dabei mindestens einen definierten Benutzer sowie einen eindeutigen Namen, d. h. eine eindeutige ID (Global Unique Identifier, GUID) besitzen. Die GUID gilt dabei über alle FIPA-Domänen hinweg, der Agent ist hierdurch im gesamten Agenten-Universum eindeutig repräsentiert. Der Agent kann dabei über die GUID an einer oder auch mehreren Adressen erreicht werden.
- Der *Directory Facilator* (DF) stellt anderen Agenten eine Art „Gelbe Seiten“ zur Verfügung. Agenten können ihre Dienste beim DF registrieren und über diesen dann abfragen, welche Dienste von anderen Agenten angeboten werden. Durch die Zugehörigkeit zu bestimmten Verzeichnissen des DF werden die Agenten logisch gruppiert; man bezeichnet eine solche Gruppe als *Agenten-Domäne*. Dabei besitzt jede Domäne genau einen DF. Die Menge aller Domänen bezeichnet man auch als *Agenten-Universum*.
- In jeder Agentenplattform gibt es genau ein *Agent Management System* (AMS), über das die Zugriffskontrolle und die Verwendung der AP geregelt wird. Das AMS beinhaltet ein Verzeichnis der logischen Agentennamen und der zugehörigen Adressen.
- Alle Agenten der AP haben auf mindestens einen *Agent Communication Channel* (ACC) Zugriff. Der ACC stellt die Standard-Kommunikationsmethode zwischen Agenten verschiedener Agentenplattformen dar.
- Der *Internal Platform Message Transport* (IPMT) dient zur AP-internen Kommunikation und wird von den FIPA Spezifikationen nicht behandelt.

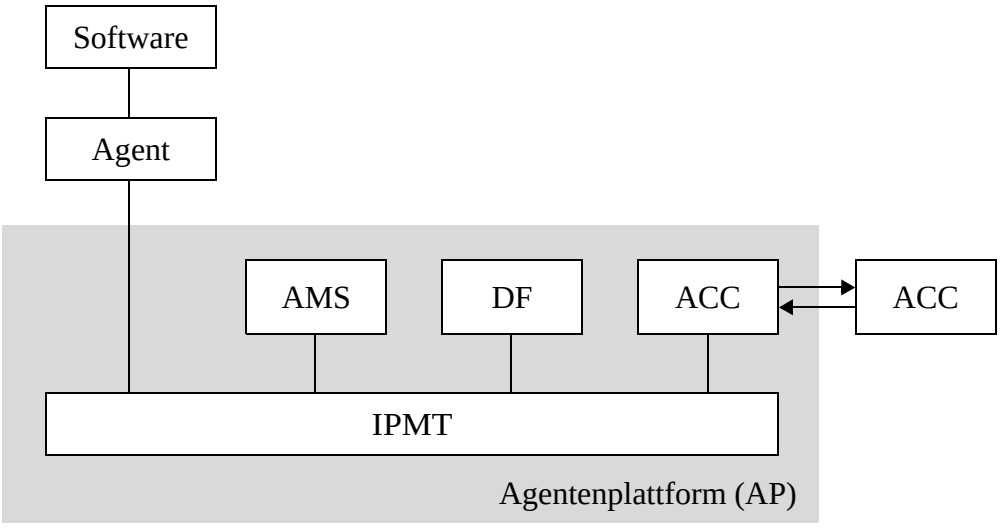

Abbildung 2-7: FIPA Agenten-Management Referenzmodell.

Das interne Design einer Agentenplattform sowie deren Implementierung wird den AP-Entwicklern überlassen und nicht von den Spezifikationen der FIPA festgelegt.

2.2.4 Kommunikationsmethoden

Damit Agenten in einem MAS miteinander kooperieren können, müssen sie primär Informationen untereinander austauschen können. Bei den hierzu verwendeten Kommunikationsverfahren unterscheidet man zwei vom Prinzip verschiedene Methoden: die Kommunikation über Shared Memory und den Austausch von Nachrichten.

2.2.4.1 Shared Memory

Bei Verwendung der Shared Memory Methode wird den Agenten ein so genanntes Blackboard zur Verfügung gestellt. Das Blackboard ist ein gemeinsamer Speicherbereich für alle Agenten, den diese lesen und beschreiben können. Schreibt ein Agent eine Information auf das Board, so ist jeder Agent für das Lesen und Interpretieren dieser Nachricht selbst verantwortlich.

Die Vorteile des Shared Memories liegen in der einfachen Handhabung und Umsetzung des Konzeptes, da es nur direkte Verbindungen vom Agenten zum Blackboard gibt. Nachteilig wirkt sich aus, dass jeder Agent alle Nachrichten lesen muss, um entscheiden zu können, ob diese für ihn relevant sind oder ob er sie ignorieren kann. Hierdurch ist auch ein Sicherheitskonzept durch entsprechende Kontrolle nur schwer integrierbar.

2.2.4.2 Nachrichtenbasierte Kommunikation

Bei der Kommunikation über Nachrichten erfolgt der Austausch der Informationen direkt zwischen den Agenten. Man unterscheidet hier entsprechend Nachrichtensender und -empfänger. In der Regel sind zur Lösung einer Aufgabe Dialoge zwischen den Agenten nötig, d. h. in diesem Fall ist ein Agent abwechselnd Sender und Empfänger. Neben dieser 1:1-Verbindung ist auch eine 1:m-Verbindung möglich, d. h. ein Agent sendet eine Nachricht an alle anderen Agenten. Diese Form der Kommunikation bezeichnet man als Broadcast.

Vorteil dieser Kommunikationsmethode ist die direkte Nachrichtenversendung von einem Agenten zum anderen, so dass keine unrelevanten Informationen abgeholt und interpretiert werden müssen. Hierdurch entsteht jedoch ein größerer Overhead, da sich zum Beispiel Agenten bei allen anderen anmelden müssen, bevor an sie Nachrichten versandt werden können.

Um Nachrichten nicht nur austauschen, sondern auch verstehen zu können, müssen die Agenten eine einheitliche Sprache sprechen. Eine in der Agentenwelt häu-

fig verwendete Kommunikationssprache ist KQML, die im folgenden Abschnitt vorgestellt wird.

2.2.4.3 KQML

Die Knowledge Query and Manipulation Language (KQML) ist ein Nachrichtenformat und ein Protokoll für den Austausch von Informationen und Wissen. KQML wurde von der External Interfaces Working Group, einem 1990 gegründetem Konsortium der Defense Advanced Research Projects Agency (DARPA) Knowledge Sharing Effort (KSE), entwickelt. Weitere Arbeitsgruppen der KSE sind die Knowledge Interchange Format (KIF) Gruppe, die Ontologie-Gruppe und die Arbeitsgruppe für Knowlegde Representations Systems Standards (KRSS).

Damit agentenbasierte Systeme überhaupt kooperieren können, müssen sie auf vier verschiedenen Stufen – zumindest in den Schnittstellen – übereinstimmen:

- im *Nachrichtentransport*, d. h. die Art und Weise, in der die Agenten Nachrichten senden und empfangen,
- in der *Kommunikationssprache*, d. h. die Bedeutung der Nachrichten muss für alle Beteiligten die gleiche sein,
- in der *Struktur*, in der die Unterhaltungen erfolgen und
- in der *Architektur*, d. h. wie die Systeme über übereinstimmende Protokolle verbunden werden.

Die KQML-Spezifikation behandelt von diesen Punkten im Wesentlichen die Kommunikationssprache.

Eine KQML-Nachricht wird in Form eines ASCII-Strings dargestellt und als *Performative* (Sprechakt) bezeichnet. In der Spezifikation werden hierzu reservierte Schlüsselwörter (Parameternamen) definiert; diese sind aber weder notwendig noch hinreichend für alle agentenbasierten Anwendungen. Agenten müssen daher nicht die komplette Menge der Performatives benutzen und können auch nicht in der Spezifikation vorkommende Nachrichten verwenden. Damit eine Applikation KQML-kompatibel ist, müssen jedoch die verwendeten reservierten Schlüsselworte mit der KQML-Spezifikation konform sein.

Der Syntax einer KQML-Nachricht ist stark an Common Lisp angelehnt. Im Gegensatz zu Lisp werden die Parameter einer Performative jedoch durch vorangestellte Schlüsselwörter indiziert und sind damit unabhängig von der Reihenfolge ihres Auftretens. Die Parameternamen müssen jeweils mit einem Doppelpunkt beginnen, gefolgt vom zugehörigen Parameterwert. Der KQML-Syntax wird in der folgenden Abbildung 2-8 in Backus-Naur-Form (BNF) definiert.

```
<performative> ::=
       (<word> {<whitespace> :<word> <whitespace> <expression>}*)

expression ::= <word> | <quotation> | <string> |
               (<word> {<whitespace> <expression>}*)

<word> ::= <character><character>*

<character> ::= <alphabetic> | <numeric> | <special>

<special> ::= < | > | = | + | - | * | / |& | ^ | ~ | _ |
              @ | $ | % | : | . | ! | ?

<quotation> ::= '<expression> | '<comma-expression>

<comma-expression> ::= <word> | <quotation> | <string> |
                       ,<comma-expression> (<word> {whitespace>
                       <comma-expression>}*)

<string> ::= „<stringchar>*“ | #<digit><digit>*“<ascii>*

<stringchar> ::= \<ascii> | <ascii>-\-|double-quote>
```

Abbildung 2-8: KQML Syntax in BNF-Darstellung.

In den beiden folgenden Tabellen werden die reservierten Schlüsselworte für die Performative-Parameter bzw. die Performative-Namen und ihre jeweiligen Bedeutungen aufgeführt. Tabelle 2-1 zeigt nur einen exemplarischen Auszug, für eine vollgeständige Liste sei auf [17] verwiesen.

Schlüsselwort	**Bedeutung**
`:content`	Eigentlicher Nachrichteninhalt
`:force`	Gibt an, ob der Absender garantieren kann, dass sich die Bedeutung der Nachricht auch in Zukunft nicht ändern wird
`:in-reply-to`	Gibt bei einer Antwort die Antwortkennung an
`:language`	Sprache, in der der Nachrichteninhalt verfasst ist
`:onthology`	Name der Ontologie, die im Nachrichteninhalt verwendet wird
`:receiver`	Empfänger der Performative
`:reply-with`	Gibt an, dass der Absender eine Antwort verlangt und enthält die zu verwendende Antwortkennung
`:sender`	Absender der Performative

Tabelle 2-1: Reservierte Performative-Parameter und ihre Bedeutungen.

Schlüsselwort	**Bedeutung**
`ask-all`	Der Absender fragt den Empfänger nach allen Antworten zur gestellten Frage
`ask-if`	Der Absender fragt beim Empfänger nach, ob der Nachrichteninhalt zu dessen Wissen passt
`ask-one`	Der Absender fragt den Empfänger nach einer Antwort zur gestellten Frage
`broadcast`	Der Absender verlangt vom Empfänger, dass dieser den Nachrichteninhalt über alle seine Verbindungen weiterleitet
`error`	Der Absender informiert den Empfänger, dass eine frühere Nachricht von diesem ein falsches Format aufgewiesen hat
`forward`	Der Absender fordert den Empfänger zur Weiterleitung der Nachricht auf
`ready`	Der Absender ist bereit zum Antworten
`register`	Der Absender meldet sich beim Empfänger an
`reply`	Der Absender antwortet auf eine frühere Nachricht des Empfängers
`tell`	Der Absender teilt dem Empfänger den Nachrichteninhalt mit
`unregister`	Der Absender meldet sich beim Empfänger ab

Tabelle 2-2: Tabelle ausgewählter, reservierter Performatives.

Die Verwendung von Nachrichten im KQML-Format soll nun anhand eines kleinen Beispiels verdeutlicht werden. Hier fragt der Absender Agent A zunächst den Empfänger Agent B, ob die Anzahl der Dreiecke eines Objektes größer ist als die eines anderen Objektes. Agent B beantwortet darauf die Frage.

```
(ask-one
     :sender agentA
     :receiver agentB
     :reply-with numberTri
     :language AgentLanguage
     :content (> (numberOfTriangles objectA)
                 (numberOfTriangles objectB)))
```

```
(reply
     :sender agentB
     :receiver agentA
     :in-reply-to numberTri
     :language AgentLanguage
     :content (true))
```

Die KQML-Spezifikation macht keine Vorgaben über den eigentlichen, im Content-Teil enthaltenen Nachrichteninhalt. Hier kann der Benutzer eine eigene Sprache oder bestehende Austauschformate verwenden. Ein häufig verwendetes Format ist das im nächsten Abschnitt beschriebene Knowledge Interchange Format.

2.2.4.4 KIF

Das Knowledge Interchange Format (KIF) ist eine formale Sprache zum Austausch von Wissen zwischen verschiedenen Computerprogrammen. Obwohl mit KIF auch das interne Wissen eines Programms beschrieben werden kann, wird das Format in der Regel nur zur Übermittlung von Wissen verwendet. Das Programm bildet also jeweils die im KIF-Format empfangene Wissensdatenbank in sein internes Format ab und umgekehrt. Ein wesentlicher Grund dafür ist, dass ähnlich dem Prinzip von Postscript KIF vor allem der unabhängigen Entwicklung eines allgemeinen Austauschformates für Wissen dient und daher beispielsweise auch nicht sonderlich effizient ist.

Bei der KIF-Syntax unterscheidet man zwei Formen: lineares und strukturiertes KIF. In linearem KIF sind alle Ausdrücke Strings bestehend aus ASCII-Zeichen. In strukturiertem KIF sind die zulässigen Ausdrücke strukturierte Objekte. KIF hat dabei seinen Syntax von Lisp geerbt. So stellt ein ASCII-String einen zulässigen Ausdruck in linearem KIF genau dann dar, wenn dieser vom Common Lisp Reader [109] akzeptiert wird und die vom Common Lisp Reader erzeugte Struktur ein zulässiger Ausdruck in strukturiertem KIFs ist.

Die strukturierten KIF-Ausdrücke bauen auf den Definitionen von `word` und `expression` auf:

```
<word> ::= ein primitives syntaktisches Objekt
<expression> ::= <word> | (<expression>*)
```

In KIF ist zum Beispiel eine Variable ein Wort, dessen erster Buchstabe ein `?` oder ein `@` ist.

Die Expressions lassen sich in vier verschiedene Typen klassifizieren:

- Bezeichnungen (terms): dienen zur Beschreibung von Objekten der betrachteten Welt
- Sätze (sentences): drücken Fakten in der jeweiligen Welt aus

- Regeln (rules): definieren zulässige Folgerungen
- Definitionen (definitions): definieren Konstanten

Hierauf aufbauend ist in der KIF-Spezifikation ein umfangreiches Vokabular für die Behandlung der unterschiedlichsten Situationen definiert. Dies beinhaltet den Umgang mit Zahlen, Listen, Mengen, Funktionen, Relationen und Metaknowledge.

Im Folgenden soll anhand einiger Beispiele die Funktionalität von KIF verdeutlicht werden. Die folgende, mathematische Funktion beschreibt beispielsweise den Absolutbetrag `abs` einer Zahl `x`:

```
(deffunction abs (?x) := (if (>= ?x 0) ?x (- ?x)))
```

Der Term (`listof` t_1 ... t_k) beschreibt eine Liste von Objekten, zum Beispiel besteht die folgende Liste aus den Objekten Computer, Monitor und der Liste mit den Objekten Tastatur und Maus:

```
(listof computer (listof keyboard mouse) monitor)
```

KIF unterscheidet zwischen Individuen (individuals) und Mengen (sets). Dabei ist ein Individuum ein Objekt, das nicht in einer Menge enthalten ist. In KIF kann man diesen logischen Ausdruck wie folgt darstellen:

```
(or (set ?x) (individual ?x))
(or (not (set ?x)) (not (individual ?x)))
```

Eine Menge wird dabei beschrieben mit dem Konstrukt (`setof` t_1 ... t_k).

Funktionen und Relationen werden in KIF durch einen Namen beschrieben, welcher dann in einem Satz an funktionaler oder relationaler Position verwendet werden kann. Syntaktisch sind Funktionen und Relationen Mengen von Listen von Objekten und werden wie folgt beschrieben:

```
(defrelation relation (?r) := ...)
(defrelation function (?f) := ...)
```

Zur Beschreibung von Wissen über Wissen (Metaknowledge) werden Ausdrücke als Objekte behandelt und geeignete Funktionen bzw. Relationen für diese bereitgestellt. Hierzu stellt man der jeweiligen Expression den Quote-Operator `'` voran. Um beispielsweise auszudrücken, dass Peter glaubt, dass der Monitor 500 Euro kostet, kann man folgendes Konstrukt verwenden:

```
(believes peter '(price monitor 500))
```

Glaubt weiter Mary alles, was Peter glaubt, so lässt sich dies wie folgt formalisieren:

```
(=> (believes john ,?p) (believes mary ,?p))
```

Dabei bedeuten die Kommata vor den Variablen, dass die jeweiligen Namen nicht wörtlich gemeint sind, d. h. hiermit kann man von den tatsächlichen Namen in der jeweiligen Wissensdatenbank abstrahieren.

Die vollständige Auflistung der einzelnen Wörter und Ausdrücke von KIF sowie deren genauen Definitionen finden sich beispielsweise in [39].

2.3 Komponententechnologie

In den vergangenen Jahren wurden besonders viele stark anwendungsbezogene Applikationen entwickelt und bei Bedarf auf andere Anwendungsfelder erweitert. Gerade im Hinblick auf die immer mehr ansteigende Rechenleistung und die zur Verfügung stehende Speichergröße bei gleichzeitig stark fallen Kosten wurde von den Softwareentwicklern häufig kaum auf eine effiziente und strukturierte Programmierung geachtet.

Auch im Visualisierungssektor war diese Arbeitsweise stark verbreitet und es entstanden dadurch große, monolithische Visualisierungssysteme, die für die gerade geforderten Bedingungen jeweils entsprechende Anpassungen oder Erweiterungen erfuhren. Im Laufe der Zeit wurden die Systeme selbst für Spezialisten aufgrund der Größe und der funktionalen Abhängigkeiten kaum noch überschaubar. Die Fehleranfälligkeit der Systeme stieg mit zunehmender Komplexität bei gleichzeitig abnehmender Wartbarkeit.

Aufgrund dieser Entwicklung existiert in der Softwareindustrie nun ein klarer Trend hin zur Komponentenorientierung, d. h. dem Aufbrechen eines großen Softwarekomplexes in viele kleine, voneinander unabhängige Teile. Wesentliche Ziele dieses Vorgehens sind die Erreichung einer größeren Flexibilität sowie die Reduzierung der Entwicklungskosten und -zeiten bei gleichzeitig gesteigerter Qualität. Die Vorgehensweise ist dabei stark von der ingenieurmäßigen Sichtweise geprägt und beruht auf der Wiederverwendung ausgetesteter, zuverlässiger Techniken und Verfahren. Die Gemeinde der Softwareentwickler lässt sich damit in zwei Lager aufspalten. Die Komponentenentwickler sind dabei für die eigentliche Entwicklung der Bausteine zuständig, während die Komponentennutzer – unterstützt durch vielseitige, in der Regel visuelle Tools – neue Anwendungen aus existierenden Komponenten generieren.

Analysiert man den Markt der Visualisierungssysteme so lässt sich feststellen, dass auch hier die moderne, komponentenorientierte Sichtweise Einzug gehalten hat. Allerdings wurden die Systeme nicht von Grund auf neu geschrieben, sondern vielmehr um die Möglichkeiten einer Entwicklungsumgebung für die Anwen-

dungsgenerierung aus entsprechenden Bausteinen ergänzt. Dabei stellt nun jedoch jedes Visualisierungssystem grundlegende Anforderungen an die funktionale und strukturelle Beschaffenheit der Komponenten. Damit ein Modul in einem System Verwendung finden kann, muss es daher in der Regel von einer oder mehreren Basisklassen abgeleitet werden. Hierdurch entsteht offensichtlich eine enge Kopplung der Komponenten mit dem verwendeten Visualisierungssystem. Bei der Übertragung auf ein anderes System sind daher umfangreiche Änderungen durchzuführen und nicht selten führen die unterschiedlichen Randbedingungen zu einer kompletten Neuprogrammierung des Moduls.

Um die beschriebenen Nachteile zu umgehen, wird in der vorliegenden Arbeit durch Verwendung der Komponententechnologie die Kopplung von Komponente und Visualisierungssystem vermieden. In diesem Abschnitt werden die benötigten Begriffe und Definitionen vorgestellt sowie die zugrunde liegenden Komponentensysteme (CORBA, COM / DCOM, Java Beans) kurz eingeführt. Eine ausführliche Beschreibung dieser Thematik findet sich in [5].

2.3.1 Definition des Komponentenbegriffes

Wie schon bei der Definition des Agentenbegriffes existieren in der Literatur und der Anwendung die verschiedensten Auffassungen und Definitionen für den Begriff der Komponente. Die wohl einfachste Definition wird in [80] gegeben:

> *Komponenten sind „[Bestand]teile eines Ganzen, die oft von gleicher Art wie dieses Ganze sind und aus denen es sich zusammensetzt oder in die es zerlegbar ist.“*

Aufgrund der Allgemeinheit dieser Definition wäre jedoch schon ein einfaches Programm, das aus verschiedenen Klassen, Funktionen oder auch nur Anweisungen aufgebaut ist, eine Komponente.

Es liegt also auf der Hand, dass der Begriff der Komponente näher spezifiziert sein muss. Eine solche Spezifikation findet sich zum Beispiel im Glossar von Microsofts MSDN Library [82]:

> *Eine Komponente ist ein Objekt, das Daten und Code einkapselt und eine wohldefinierte Menge von öffentlich verfügbaren Diensten bereitstellt.*

Aber auch diese Aussage lässt viele Fragen offen. Zum Beispiel wird hierdurch nicht geklärt, wie die Dienste anderen Objekten zur Verfügung gestellt werden sollen (Interfaces) und wie diese Schnittstellen verwaltet und zum Zusammenfügen von Komponenten genutzt werden können.

Eine im Kontext dieser Arbeit geeignete Definition der Komponente wird von Barthel [5] formuliert, der die 1996 von einer Arbeitsgruppe der ECOOP (European Conference on Object-Oriented Programming) gegebene Begriffsbildung um Aussagen über die Verwaltung von Schnittstellenbeschreibungen erweitert:

> *Eine Softwarekomponente ist ein Stück Software, das über eine standardisierte, vertraglich festgelegte Schnittstelle (Interface) verfügt. Die Definition von Interfaces sowie ihre Verwaltung wird durch ein Komponentensystem (Repository) geregelt. Die unabhängige Verwendung von Komponenten in einem Anwendungssystem wird durch die im Komponentensystem verwalteten Informationen über die Interfaces realisiert.*

Diese Definition verlangt also nicht nur ein gekapseltes Objekt mit fest definierten Schnittstellen, sondern auch das Vorhandensein eines Komponentensystems, das die Interfaces verwaltet und anderen Bausteinen zur Verfügung stellt. Bevor nun näher auf diese Systeme eingegangen wird, sollen zunächst die wichtigsten Vorteile der Komponenten selbst skizziert werden.

2.3.2 Eigenschaften der Komponententechnologie

Durch die Einführung der Komponententechnologie ergeben sich aufgrund der Wiederverwendbarkeit einzelner Softwaremodule vielfältige Vorteile gegenüber der herkömmlichen Programmierweise:

- *Trennung zwischen Komponentenentwickler und -benutzer*: Lediglich für den Entwickler ist die Sicht auf den zugrunde liegenden Programmcode nötig und auch nur möglich. Der Benutzer einer Komponente sieht diese als eine Art „Black Box“, von der er lediglich Informationen wie Funktionalität und Schnittstellen kennen muss. Allein mit diesen Informationen kann er eigene Anwendungen aus bestehenden Baussteinen zusammensetzen. Verwendet er zusätzlich noch ein visuelles Tool zur Erzeugung der Applikationen, so sind hierfür noch nicht einmal Programmierkenntnisse erforderlich.
- *Geringere Fehleranfälligkeit*: Bei der Erstentwicklung einer Komponente wird diese in der Regel entsprechenden Evaluierungsphasen unterzogen. Durch die Kompaktheit und Abgeschlossenheit der Module sind diese Testläufe sehr effizient und aussagekräftig. Bei der Wiederverwendung einer Komponente ist diese bereits evaluiert, so dass in diesem Fall direkt ein stabiler Baustein eingesetzt werden kann. Auch werden die Module ständig erweitert und weiter verbessert. Die beschriebenen Prozesse führen somit zu einer gesteigerten Qualität der einzelnen Bausteine und damit auch des aus ihnen zusammengesetzten Gesamtsystems.
- *Kürzere Entwicklungszeiten bei gleichzeitig gesenkten Kosten*: Aufgrund der Wiederverwendbarkeit von Bausteinen lässt sich der Entwicklungsbedarf bei

vielen Anwendungen auf einige wenige Komponenten reduzieren. Die restlichen benötigten Module können aus dem Pool eigener oder zugekaufter Komponenten einfach entnommen und in die Anwendung integriert werden. Durch die somit offensichtlich schnellere Entwicklung wird entscheidend zur Senkung der entsprechenden Kosten beigetragen. Hinzu kommt noch die höhere Zuverlässigkeit der einzelnen Bausteine bei der Wiederverwendung, so dass hier auch die normalerweise erforderlichen Testphasen reduziert werden und teilweise sogar ganz entfallen können.

- *Geringere Komplexität*: Jedes Modul in einer aus Komponenten zusammengesetzten Anwendung hat eine scharf umrissene Funktionalität und stellt sich dem Benutzer als „Black Box“ dar. Damit lassen sich auch komplexe Applikationen übersichtlich und effizient entwickeln sowie Designfehler leichter erkennen.
- *Hohe Flexibilität*: Einzelne Komponenten in einem Gesamtsystem lassen sich unabhängig von anderen Bausteinen austauschen oder aktualisieren, so dass Applikationen schnell an sich ändernde äußere Bedingungen anpassbar sind. Zusammen mit der geringeren Komplexität ergeben sich so Vorteile im Hinblick auf die Systemwartung und -erweiterung.

Die obige Liste unterstreicht eindrucksvoll die Vorteile komponentenorientierter Entwicklung. Grundlage hierfür bildet zunächst die Auswahl eines geeigneten Komponentensystems.

2.3.3 Komponentensysteme

Die in Abschnitt 2.3.1 gegebene Definition einer Komponente fordert als zentralen Punkt das Vorhandensein eines Komponentensystems, das zur Verwaltung und Administration der einzelnen Interfaces dient.

Von der Funktionsweise her betrachtet, lässt sich ein Komponentensystem gut mit dem Client-Server-Modell vergleichen. Dabei bietet eine Komponente einen bestimmten Dienst über ihre Schnittstellen anderen Bausteinen an und fungiert damit als Server. Andere Komponenten verwenden den zur Verfügung gestellten Dienst und können somit als Client betrachtet werden. Da aber auch die Clients wiederum jeweils ihren eigenen Service anderen Bausteinen anbieten, sind diese gleichzeitig auch Server. Die Unterscheidung in Client- bzw. Server-Bausteine ist also nur auf einen bestimmten Dienst bezogen möglich. Betrachtet man dagegen ein Gesamtsystem, so sind die Bausteine in der Regel als gleichberechtigte Partner ähnlich einer Peer-to-Peer-Struktur anzusehen.

Die eigentliche Verbindung zwischen den Bausteinen, d. h. die Kommunikation zwischen ihnen, übernimmt das Komponentensystem, indem es eine geeignete Kommunikationsinfrastruktur zur Verfügung stellt.

In den folgenden Abschnitten werden die drei zurzeit gängigsten Komponentensysteme vorgestellt: CORBA, COM / DCOM und Java Beans.

2.3.3.1 CORBA

Die Common Object Request Broker Architektur (CORBA) ist die praktische Umsetzung der von der Object Management Group (OMG) vorgeschlagenen Object Management Architecture (OMA), die sich ihrerseits aus Komponenten, Schnittstellen und Protokollen zusammensetzt.

Die OMG wurde 1989 von 11 Firmen, unter ihnen renommierte Namen wie 3Com, American Airlines, Canon, HP, Philips und Sun, gegründet. Heute gehören ca. 800 Firmen zu diesem Konsortium. Microsoft bildet hier eine Ausnahme, da diese ihre Eigenentwicklung COM / DCOM (siehe Abschnitt 2.3.3.2) als Standard durchsetzen möchten.

Gründungsziel war die Definition eines von der eingesetzten Programmiersprache unabhängigen Standards für komponentenbasierte Software. Die Aufgaben der OMG beinhalten weiter die Etablierung von Industrierichtlinien und detaillierten Objektmanagementspezifikationen, um ein allgemeines Framework für die Anwendungsentwicklung zur Verfügung zu stellen [89].

Abbildung 2-9 zeigt schematisch die Object Management Architecture.

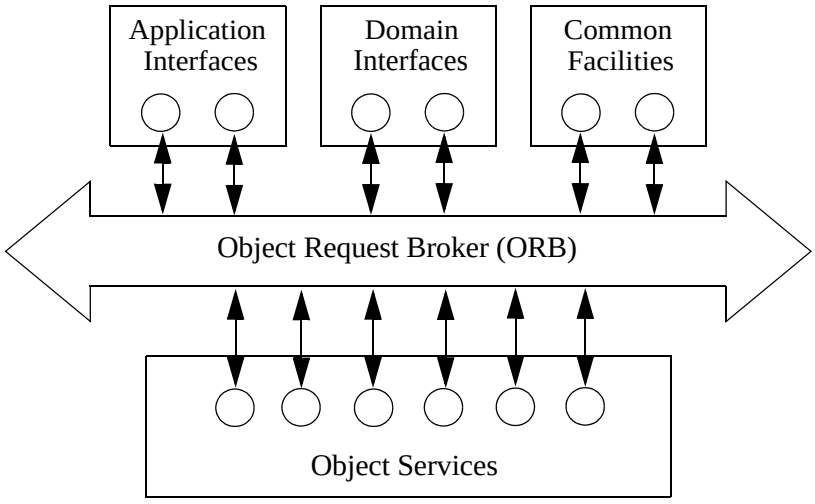

Abbildung 2-9: Die Object Management Architecture (OMA).

Zentraler Bestandteil dieser Architektur ist der Object Request Broker (ORB). Aufgabe des ORBs ist die Zurverfügungstellung einer transparenten Kommunikationsinfrastruktur. Mit Hilfe des ORB können die Objekte unabhängig von Plattform und ihrer Implementierungssprache miteinander kommunizieren. Die Com-

mon Object Request Broker Architektur (CORBA) definiert die Programmierschnittstellen zur OMA-ORB-Komponente.

Die OMG Interface Definition Language (IDL) stellt dabei eine standardisierte Möglichkeit zur Beschreibung der Schnittstellen zu CORBA-Objekten zur Verfügung. Der IDL-Syntax ist stark an den ANSI-Standard der Programmiersprache C++ angelehnt. Durch die IDL wird keine neue Programmiersprache definiert, sie ist rein deklarativ und dient lediglich zur einheitlichen und konsistenten Interfacebeschreibung. Damit wird die IDL programmiersprachen- und plattform-unabhängig. Erst durch den IDL-Compiler erfolgt ein Mapping in die in der Applikation verwendeten Programmiersprache. Unterstützt werden hier beispielsweise C, C++, Cobol und Java.

Die Object Services ergänzen den ORB durch allgemeine, systemabhängige Funktionalitäten, die von jeder auf verteilten Objekten basierenden Applikation verwendet werden können. Beispiele für solche Dienste sind Naming, Events, Persistence, Transactions, Relationships, Licensing, Query, Properties und Security.

Die Common Facilities sind allgemeine, endbenutzer-orientierte Funktionalitäten, die in den meisten Anwendungsdomänen anwendbar und entsprechend konfigurierbar sind. Beispiele hierfür sind die Printing oder Database Facilities.

Die Domain Interfaces sind im Vergleich zu den Common Facilities weniger allgemein und stellen für den jeweiligen Anwendungsbereich (zum Beispiel Finanzen, Telekommunikation, Transportwesen usw.) spezifische Schnittstellen zur Verfügung.

Die Application Interfaces schließlich fassen die nicht-standardisierten und applikationsabhängigen Schnittstellen zusammen.

Die Kommunikation entspricht dem Client-Server-Modell. Der Client startet durch den Zugriff auf eine Schnittstelle des ORBs eine Anfrage (Request), um die Funktionalität eines Dienstes eines Server-Objektes zu nutzen. Mittels Response bzw. Reply wird dann die Antwort des Servers an den Client zurückgeschickt. Dabei können Client und Server sowohl auf der gleichen Maschine als auch verteilt in einem Netzwerk residieren. Im ersten Fall wird ein gemeinsamer ORB benutzt, ansonsten werden zwei ORBs verwendet. Der Client muss dabei nur seine eigenen Interfaces kennen, auf der Server-Seite kapseln die Schnittstellen die eigentliche Implementierung der Methoden.

CORBA stellt zwei verschiedene Kommunikationsmöglichkeiten zur Verfügung, wodurch der Aufruf eines Dienstes, der von einem Server-Objekt angeboten wird, sowohl statisch als auch dynamisch erfolgen kann. Beim statischen Aufruf verwendet der Client ein Stellvertreterobjekt (stub) für die eigentliche Implementierung, das den Methodenaufruf an den ORB weitergibt. Der Server verwendet dann

den ihm vom ORB zugeleiteten Stellvertreter, den so genannten Skeleton, zum eigentlichen Aufruf des Dienstes. Stubs und Skeleton werden bei der Übersetzung der IDL-Quellen jeweils automatisch erzeugt. Beim dynamischen Aufruf eines Services kann der Client die aufzurufende Methode und deren Parameter während der Laufzeit festlegen. Hierzu verwendet der Client das so genannte Dynamic Invocation Interface (DII) und der Server das dazu gehörende Gegenstück, das Dynamic Skeleton Interface (DSI).

2.3.3.2 COM / DCOM

Das Komponentenmodell COM / DCOM ist ein Unternehmensstandard von Microsoft und entstand aus einer Reihe von Einzelentwicklungen, die die bessere Abstimmung von Office-Anwendungen und Betriebssystem zum Ziel hatten.

Im Laufe der Zeit – vor allem bei der Einführung neuer Microsoft Produkte – wurden dabei die Begriffe sowohl im Hinblick auf ihre Bedeutung als auch auf den von ihnen beschriebenen Umfang zum Teil neu orientiert, was eine exakte Abgrenzung und Einordnung erschwert.

Am Beginn der Entwicklung stand die Einführung der Object Linking and Embedding (OLE) Technologie. Diese diente vor allem zur Erzeugung von Verbunddokumenten. Mit dieser Technik wurde zum Beispiel das Kopieren und Einfügen (Cut and Paste) zwischen verschiedenen Office-Anwendungen ermöglicht. OLE wurde dann um das allgemeinere Komponentenmodell COM (Component Object Model) zu OLE2 erweitert.

Der Einsatz der COM-Technologie war jedoch nicht nur auf Verbunddokumente beschränkt, sondern erlaubte vielmehr das Zusammenspiel beliebiger Softwareanwendungen. Die Bezeichnung OLE2 als Obermenge von COM war somit eher unglücklich gewählt und Microsoft führte daher 1996 den Begriff ActiveX ein, um den Inhalt des OLE-Begriffes wieder auf den ursprünglichen Umfang zurückzuführen. Unter ActiveX lassen sich alle auf COM basierenden Techniken zusammenfassen.

Da COM die Möglichkeiten zur verteilten Kommunikation fehlten, erfolgte mit der Einführung von Windows NT die Erweiterung auf das Distributed Component Object Model (DCOM). COM bzw. DCOM ist heute das den Windows-Markt beherrschende Komponentenmodell, da es in allen aktuellen Microsoft Betriebssystemen integriert ist. Nachteilig wirkt sich jedoch die naturgemäß starke Ausrichtung auf die Microsoft Windows-Welt aus. Durch eine Kooperation mit der Software AG sind jedoch auch Portierungen auf andere Plattformen entstanden, allerdings mit einer wesentlich geringeren Verbreitung als zum Beispiel bei CORBA.

.

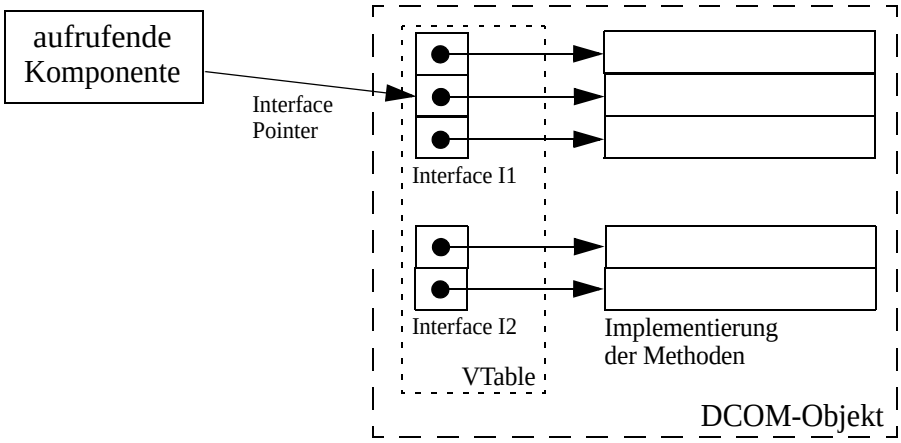

Abbildung 2-10: Aufruf einer Methode über eine virtuelle Funktionstabelle.

DCOM ist wie auch CORBA sprachunabhängig und basiert auf einem binären Standard, über den die Dienste der entsprechenden Komponenten angesprochen werden können. Möchte eine Client-Komponente eine bestimmte Funktionalität eines DCOM-Objektes aufrufen, so erhält diese zunächst einen Pointer auf das gewünschte Interface. Dieser Interface Pointer verweist auf die Interface Function Table des DCOM-Objektes, eine virtuelle Funktionstabelle (VTable), in der wiederum für jede Methode der Schnittstelle Zeiger auf die eigentlichen Implementierungen abgelegt sind (siehe Abbildung 2-10). DCOM-Objekte werden dabei über eine eindeutige 128 Bit große Identifikationsnummer (GUID – Globally Unique Identifier) angesprochen, die aus der Prozessornummer und dem Zeitstempel eindeutig generiert werden. Entsprechend werden auch die einzelnen Interfaces über eine eindeutige ID (IID – Interface Identifier) angesprochen.

Die Mindestanforderung an ein DCOM-Objekt ist das Vorhandensein der Schnittstelle `IUnknown`, bestehend aus den Methoden `QueryInterface`, `AddRef` und `Release`. Der Methodenaufruf `QueryInterface` liefert dem Client-Objekt als Rückgabewert einen Pointer auf ein Interface der angesprochenen Server-Komponente. Die beiden anderen Methoden dienen zur Verwaltung der DCOM-Objekte über einen Referenzzähler. Beim Aufruf eines Server-Dienstes wird beim zugehörigen Objekt über `AddRef` der Zähler erhöht. Wird der Dienst nicht mehr benötigt, so wird das Server-Objekt mittels `Release` wieder freigegeben und der Zähler entsprechend erniedrigt. Ist der Referenzzähler auf Null, so kann das Server-Objekt zerstört werden.

Interfaces können in DCOM nicht mehrfach vererbt werden, stattdessen kann eine Komponente andere Komponente enthalten. Zur Realisierung stehen dem Anwender zwei verschiedene Möglichkeiten zur Verfügung:

- Containment / Delegation: Die eigentlichen Schnittstellen der inneren Komponenten werden nicht nach außen gezeigt, sondern in der Liste der Interfaces der äußeren Komponente aufgenommen. Wird ein Dienst einer inneren Komponente aufgerufen, so ruft der Client das Interface der äußeren Komponente auf. Diese ruft wie ein normales Client-Objekt dann die Methode der inneren Komponente auf.
- Aggregation: Die Interfaces der inneren Komponenten werden direkt nach außen geschleift. Die Aggregation muss explizit von der entsprechenden inneren Komponente unterstützt werden.

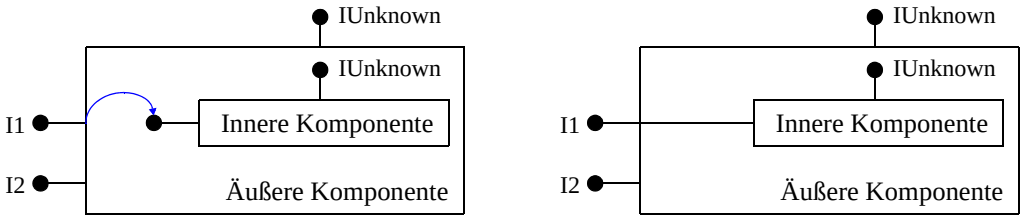

Abbildung 2-11: Containment / Delegation (links) und Aggregation (rechts).

Der Lifecycle, d. h. die Versionierung, einer DCOM-Komponente erfolgt nach festgelegten Regeln. Eine neue Version eines DCOM-Objektes entsteht jeweils, wenn die Komponente interne Modifikationen oder Funktionalitätserweiterungen erfährt. Dabei dürfen keine Schnittstellen der Vorversion verändert werden, sondern für neu hinzugekommene Methoden müssen entsprechend neue, zusätzliche Interfaces generiert werden. Hierdurch wird garantiert, dass die Ersetzung einer DCOM-Komponente durch eine neue Version in einem Softwaresystem keine komponentenexternen Anpassungen nötig macht. Da auch keine Funktionalitäten während des Lifecycles wegfallen dürfen, wächst die Größe der einzelnen Komponenten in der Regel mit jeder neuen Version an. In der Praxis wird daher zu bestimmten Zeitpunkten ein so genannter Reengineering-Prozess durchgeführt, bei dem eine neue, kleinere Komponente mit anderen Methoden generiert wird. Das neu erzeugte DCOM-Objekt erhält gleichzeitig auch einen anderen Namen, um es eindeutig von der Vorgänger-Komponente zu unterscheiden. Ab diesem Zeitpunkt werden in der Regel alle Wartungs- und Erweiterungsarbeiten an der alten Komponente eingestellt.

Für die Kommunikation zwischen den einzelnen DCOM-Objekten zeichnet sich die so genannte Windows Registry verantwortlich, eine zentrale Datenbank, in der Systeminformationen wie zum Beispiel installierte Softwarepakete, Hardwarekomponenten, Device-Treiber usw. abgelegt sind. Die Registry entspricht damit dem Repository der Komponenten-Definition.

2.3.3.3 Java Beans

Die Java Beans-API ist eine von Sun Microsystems entwickelte Softwarekomponentenarchitektur für Java ([112], [113]). Eines der Hauptziele bei der Entwicklung von Java Beans war es, neben den Komponentenentwicklern auch die Komponentenbenutzer mit Hilfe von grafischen Entwicklungswerkzeugen zu unterstützen. Dies spiegelt sich auch in der von Sun gegebenen Definition der Java Bean – oder kurz auch nur Bean – wider:

> *A Java Bean is a reusable software component that can be manipulated visually in a builder tool.*

Der Einsatz der angesprochenen Tools zur visuellen Manipulation einer Java Bean ist dabei aber nur optional, d. h. auf ihren Einsatz kann auch verzichtet werden. In diesem Fall kann eine Java Bean als gewöhnliche Java-Klassenbibliothek verwendet werden. Die zugrunde liegende Java Beans-Spezifikation ist daher klassenbasiert und legt lediglich einige Design Patterns (also eine Menge von Regeln) fest, die eine Standard-Java-Klasse einhalten muss, um zu einer Java Bean zu werden.

Der Einsatz der Java Beans-Technologie ist nicht auf bestimmte Applikationen oder Plattformen beschränkt. So ist es zum Beispiel möglich, Komponenten beliebiger Größe und Komplexität zu entwickeln. Analog zur Programmiersprache Java sind auch die Java Beans plattformunabhängig, d. h. eine Java Bean bietet auf allen Plattformen die gleiche Funktionalität. Zentrale Voraussetzung für den Einsatz ist dabei das Vorhandensein einer Java-Implementierung in Form der Java Virtual Machine (JVM). Die JVM ist im Gegensatz zu den entwickelten Java-Programmen plattformabhängig, um die jeweiligen spezifischen Features unterstützen zu können. Der Entwickler einer Bean muss sich somit nicht um die speziellen Begebenheiten auf unterschiedlichen Hardwareplattformen kümmern, da die JVM jeweils eine entsprechende Funktionalität auf allen Systemen gewährleistet.

Der Zugriff auf die Dienste einer Java Bean erfolgt über ihre öffentliche Schnittstelle, die sich in Methoden, Properties (Eigenschaften) und Events (Ereignisse) unterteilen lässt. Die Schnittstellendefinition erfolgt in der BeanInfo-Klasse. Diese Information wird zum Beispiel von den Entwicklungswerkzeugen genutzt und kann manuell vom Entwickler implementiert, durch Introspection automatisch generiert oder unter Verwendung der von der OMG definierten IDL-Beschreibungssprache festgelegt werden. Durch die Verwendung der IDL ist es Java Beans-Komponenten möglich, Dienste von CORBA-Komponenten zu nutzen, aber auch ihre Dienste CORBA-Objekten zur Verfügung zu stellen.

Die Events stellen den Kommunikationsmechanismus der Java Beans dar. Dabei bezeichnet man eine Komponente, die eine Nachricht gekapselt in einem Event Object an andere Komponenten verschickt, als Event Source und den Ereignis-

empfänger als Event Listener. Für jeden Event-Typ existieren eindeutige Eventhandling-Methoden, die in so genannten Event Listener-Interfaces definiert werden. Eine Event Source muss Registrierungsmethoden zum An- und Abmelden von Event Listenern definieren, um entsprechend Nachrichten an diese Listener verschicken zu können. Den Zusammenhang zwischen den soeben eingeführten Begriffen veranschaulicht Abbildung 2-12.

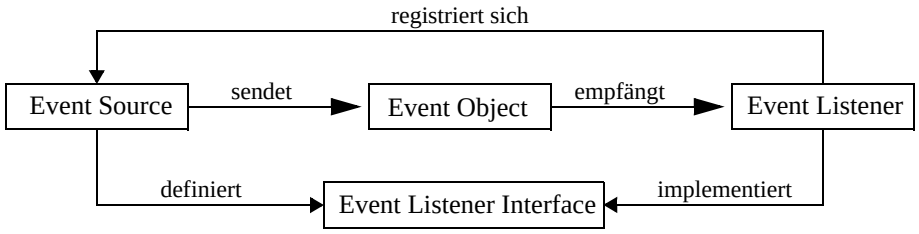

Abbildung 2-12: Events in der Java Beans-Architektur.

Die Java Beans-Spezifikationen definieren dabei die Design Patterns für die benötigten Registrierungs-Methoden:

public void add<EventListenerTyp> (<EventListenerTyp> Listener);
public void remove<EventListenerTyp> (<EventListenerTyp> Listener);

Die Properties bestimmen die Eigenschaften und das Verhalten einer Java Bean-Komponente. Der Zugriff auf die einzelnen Properties erfolgt über entsprechende get- bzw. set-Methoden. Das Vorhandensein dieser Methoden definiert gleichzeitig die zugehörige Zugriffsart: ist beispielsweise nur eine get-Methode implementiert, so ist die Property read-only. Entsprechend lassen sich write-only- und read / write-Zugriffe realisieren. In den Java Beans-Spezifikationen sind die zugehörigen Design Patterns für die get- und set-Methoden definiert. Der beiden Funktionen gemeinsame Name ist gleichzeitig der Name der Property selbst:

public void set<Name> (<PropertyTyp> Wert);
public <PropertyTyp> get<Name> ();

Die Properties lassen sich in vier verschiedene Gruppen einteilen: Simple Properties, Indexed Properties, Bound Properties und Constrained Properties. Während die Simple Properties jeweils nur einen einzigen Wert symbolisieren, der dann über entsprechende Methoden gelesen oder geschrieben werden kann, kann mit Hilfe der Indexed Properties auch mit Vektoren gearbeitet werden, auf die über einen Index zugegriffen werden kann. Die Design Patterns sind in diesem Fall entsprechend erweitert:

public void set<Name> (int index, <PropertyTyp> value);
public <PropertyTyp> get<Name> (int index);

public void set<Name> (<PropertyTyp> values[]);
public <PropertyTyp>[] get<Name> ();

Oftmals ist es aber auch nötig, andere Komponenten von der Änderung eines Wertes einer Property zu unterrichten. Hierzu feuern die Bound Properties einen PropertyChangeEvent an alle registrierten (PropertyChange)Listener, so dass diese dann auf die bereits erfolgte Wertänderung reagieren können. Im Gegensatz hierzu werden bei Constrained Properties die Listener bereits über die Absicht einer Wertänderung informiert. Diese können dann ein Veto mittels einer PropertyVetoException einlegen. Nur wenn keiner der Listener ein solches Veto ausspricht, ist die Wertänderung zulässig und darf entsprechend ausgeführt werden.

Java ist eine dynamische Programmiersprache. Ein Objekt kann bei gegebener Referenz auf ein anderes Objekt viele Informationen, zum Beispiel dessen Methoden, die Parameter der Methoden und den Rückgabetyp, automatisch herausfinden. Damit kann ein Objekt ein anderes Objekt instanziieren und während der Laufzeit einen entsprechenden Methodenaufruf generieren. Im Allgemeinen bezeichnet man die beschriebene Sammlung aller Informationen über Properties, Events und Methoden einer Komponente mit Introspection, unterschieden nach automatischer und expliziter Introspection. Die automatische Introspection analysiert unter Verwendung der Reflection-API eine Klasse und deren Methoden und sammelt die gefundenen Informationen über alle öffentlichen Properties, Events und Methoden in der entsprechenden BeanInfo-Klasse. Möchte der Entwickler einer Komponente jedoch nur eine Teilmenge dieser Informationen öffentlich machen, so kann er die BeanInfo-Klasse auch selbst angeben. Bei dieser expliziten Introspection wird keine Analyse der Klasse mittels der Reflection-API durchgeführt, sondern stattdessen die in der vom Entwickler generierten BeanInfo-Klasse gespeicherten Informationen verwendet.

Neben dem direkten Ansprechen einer Property über die BeanInfo-Klasse steht noch das so genannte Customizing zur Verfügung. Hiermit können die Eigenschaften einer Komponente interaktiv über einen GUI-Dialog verändert werden, was gerade bei visuellen Entwicklungswerkzeugen zum Einsatz kommt. Analog zur Introspection können die Properties einer Java Bean entweder alle (Property Sheet) oder selektiv (Customizer) dargestellt werden. Eine Java Bean besitzt so genannte Property Sheets, die für den Zugriff auf die Properties der Komponente jeweils einen Property-Editor zur Verfügung stellen. Die Funktionalität eines Customizers (zuständig für die gesamte Bean) geht über die der Property Sheets hinaus und erlaubt beispielsweise auch die Berücksichtigung von Abhängigkeiten zwischen einzelnen Eigenschaften. Um einen solchen Customizer zur Verfügung zu stellen, muss der Entwickler eine GUI-Komponente (implementiert das Customizer-Interface) und eine entsprechende BeanInfo-Klasse (liefert die Klasse des Customizers zurück) implementieren.

Oftmals ist es wünschenswert, den Zustand einer Komponente – zum Beispiel nach dem Customizing – abspeichern zu können. Diese Sicherung des internen Zustands eines Objektes und die damit mögliche spätere Zustandsrestaurierung bezeichnet man als Persistenz. In Java sind zur Realisierung der Persistenz zwei Varianten vorgesehen: Serialization und Externalization. Damit eine Komponente serialisierbar wird, muss diese nur als solche gekennzeichnet werden, indem sie das Interface java.io.Serializable implementiert. Die eigentliche Serialisierung erfolgt dann automatisch durch die Java Virtual Machine, dabei wird der momentane Zustand einer Komponente als binärer Stream (enthält eine eindeutige Klassen-ID, den Klassennamen und für alle Attribute Namen, Typ und Wert) gesichert. Ähnlich dem Customizing können die Informationen auch selektiv abgespeichert werden, diesen Prozess bezeichnet man als Externalization. Der Komponentenentwickler muss hierzu das Format und die zu speichernde Information selbst bestimmen und dafür Sorge tragen, dass der geschriebene Bytestream bei der Deserialisierung auch wieder eingelesen werden kann.

Serialisierte Beans können in einer Archiv-Datei (JAR-File) komprimiert gespeichert werden. Damit wird zum Beispiel der einfache Download von Beans in einem WWW-Browser ermöglicht.

Die Java Beans-Spezifikation definiert eine äußerst flexible Komponentenarchitektur, die lediglich auf einer Reihe von Design Patterns beruht. Allein durch das Einhalten dieser Design Patterns wird eine gewöhnliche Java-Klasse zu einer Java Bean. Zudem kann die Baussteinentwicklung völlig unabhängig von anderen Komponenten erfolgen, da nicht von vorgegebenen Basisklassen abgeleitet werden muss. Durch die Verwendung der IDL bei der Schnittstellenspezifikation ist zudem die problemlose Anbindung von CORBA-Diensten möglich.

2.3.4 Visual Prototyping

Unter Visual Prototyping versteht man das interaktive Zusammensetzen einer Applikation aus verschiedenen Modulen mit Hilfe eines grafischen Entwicklungswerkzeuges. Hierzu versteht man unter einer Anwendung ein aus einzelnen Modulen bestehendes Netzwerk. Ein Modul kann Daten empfangen, verarbeiten und ausgeben. Für die Verwendung im Netzwerk besitzt es daher entsprechende Eingangs- bzw. Ausgangskanäle. Die einzelnen Bausteine werden dem Benutzer über eine Modulbibliothek zur Verfügung gestellt. Ein Datenfluss zwischen zwei Modulen entsteht durch einfaches Verbinden eines Ausgangskanals des einen Moduls mit einem Eingangskanal eines anderen (die Kompatibilität der Kanäle sei hier vorausgesetzt). Zusätzlich kann der Benutzer in der Regel die einzelnen Parameter der Bausteine über einen entsprechenden GUI-Dialog anzeigen und an seine Bedürfnisse anpassen.

Die meisten Visualisierungsanwendungen weisen eine hohe und oft nur schwer überschaubare Komplexität und Parametrisierung auf. Daher bieten viele Visualisierungssysteme neben einer Programmierschnittstelle zusätzlich eine Visual Prototyping-Umgebung an, d. h. einen interaktiven Editor für den Aufbau und die Datenflussbeschreibung einer Anwendung. Bei den existierenden Visualisierungssystemen sind jedoch die Module eng mit dem zugrunde liegenden System gekoppelt, d. h. jedes Modul muss für die Integration zunächst systemtechnische Vorgaben (Funktionen, Klassen) erfüllen. Dies erzeugt einen nicht unerheblichen zusätzlichen Aufwand bei der Einarbeitung in die Struktur und die softwaretechnischen Gegebenheiten des Systems. Da zudem jedes Visualisierungssystem eine eigene, nicht mit anderen Systemen kompatible Architektur besitzt, ist die übergreifende Nutzung ein und desselben Moduls in mehreren Visualisierungssystemen in der Regel nicht ohne weitreichende, manuelle Änderungen möglich. Jedes System weist zugleich verschiedene Stärken und Schwächen auf. Die Benutzer sind daher gezwungen, entweder ihre Algorithmen an mehrere, zum Teil zudem recht kostspielige Systeme anzupassen oder sich für ein einziges System zu entscheiden, an welches sie dann ihre Algorithmen anpassen und etwaige Nachteile in Kauf nehmen.

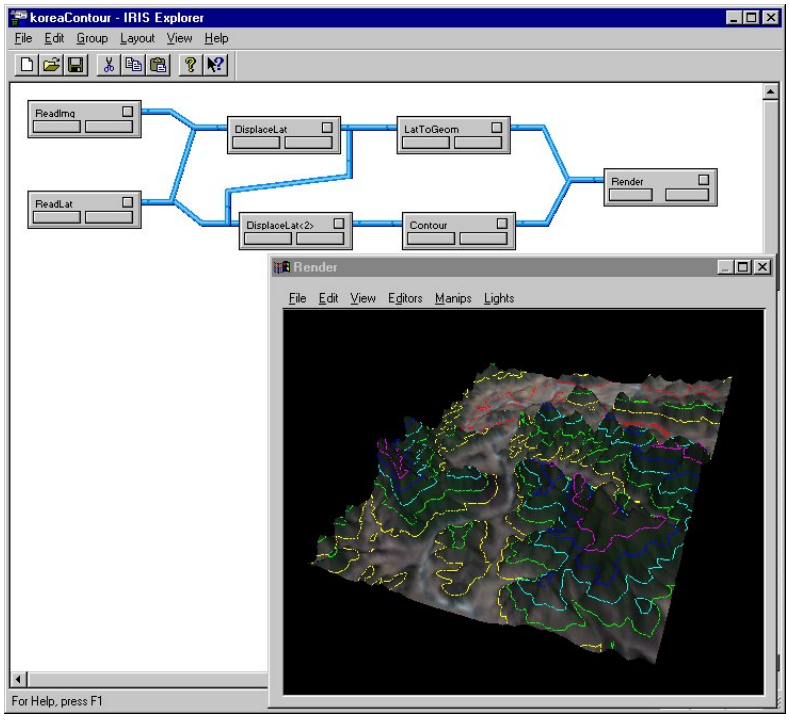

Abbildung 2-13: Visual Prototyping in IRIS Explorer: Visualisierung von Höhenlinien.

3

Kontextsensitive Visualisierung

3.1 Evaluierung existierender Visualisierungsparadigmen

3.1.1 Visualisierungsbibliotheken

Für die Kommunikation zwischen Grafikhardware und Software zeichnen sich verschiedene Multimediabibliotheken verantwortlich, wobei (auf Windows-Ebene) DirectX und OpenGL die heutzutage gebräuchlichsten APIs (Anwendungsprogrammierschnittstellen) sind. Dabei können moderne Grafikkarten beide Schnittstellen bedienen, jeweils mit mehr oder weniger Hardwareunterstützung. In der Entwicklergemeinde gibt es dabei Verfechter sowohl der einen als auch der anderen Technologie.

Neben diesen relativ hardwarenahen Low-Level-Schnittstellen gibt es noch Visualisierungsbibliotheken, die direkt auf OpenGL und / oder DirectX aufsetzen, dem Programmierer jedoch die Programmierung durch ihre High-Level-API erleichtern. Im Rahmen dieser Arbeit wird das Visualization Toolkit (VTK) und Sun's Java 3D-API betrachtet.

3.1.1.1 DirectX, Direct3D

Microsoft DirectX ([21],[81],[82],[106]) ist eine Ende 1995 von Microsoft eingeführte Multimediabibliothek, auf der viele Windows-Applikationen zur Generierung von Grafiken, Videos, 3D-Animationen oder auch Surround Sound aufsetzen. Hauptziel von DirectX war es, die damals bestehenden Nachteile von Win-

dows-Programmen gegenüber MS-DOS-Anwendungen zu beseitigen und gleichzeitig die Vorteile von Windows gegenüber MS-DOS auszunutzen. Während unter MS-DOS ein direkter Zugriff auf die Hardware mangels zentraler Handwareverwaltung zwingend erforderlich war, erhält bei Multitasking-Betriebssystemen wie Windows nur das Betriebssystem exklusiven Zugriff auf die Hardware. Die Abstraktions-Layer führen zu einem sehr beschränkten und langsamen Zugriff auf Grafik- und Sound-Hardware. Microsoft entwickelte aus diesem Grund eine API, welche Entwicklern direkten und gleichzeitig hardwareunabhängigen Zugriff auf die Hardware erlaubt, um vor allem Spiele mit akzeptablen Geschwindigkeiten laufen zu lassen.

DirectX ist zurzeit erhältlich für die gängigen Microsoft Plattformen Windows 95/98/Me, Windows NT, Windows 2000 und Windows XP. Die aktuelle Versionsnummer ist 9.0b für Windows 95/98/Me/2000/XP bzw. 3.0 für Windows NT (hier stehen daher moderne DirectX-Funktionalitäten zum Teil nicht oder nur eingeschränkt zur Verfügung).

Aus der Sicht des Programmierers ist DirectX eine Hardware-Schnittstelle, die den Zugriff auf die Hardware in einer entsprechenden API kapselt. Diese Kapselung erlaubt damit in Bezug auf die Windows-Welt eine hardwareunabhängige Programmierung. Der Benutzer installiert zum Beispiel beim Wechsel der Grafikkarte einfach den zugehörigen Treiber, DirectX-Konformität desselben natürlich vorausgesetzt. Die Applikationen bleiben dabei unverändert, da sich für sie die Schnittstelle durch den Austausch des Treibers nicht verändert.

Die DirectX-API ist dabei in zwei Klassen unterteilt: die DirectX Foundation-Ebene und die DirectX Media-Ebene. Diese APIs ermöglichen es Programmen, auf viele Hardwaregeräte des Computers direkt zuzugreifen.

Die DirectX Foundation-Ebene ermittelt automatisch die Hardwarefunktionen des Computers und stellt dann die Programmparameter entsprechend ein. Dadurch können unter Windows Multimediaanwendungen auf jedem Computer ausgeführt werden, und es wird gleichzeitig sichergestellt, dass die Anwendungen die vorhandene Hardware entsprechend deren Leistungsmöglichkeiten nutzen. Die DirectX Foundation-Ebene enthält eine einzelne Gruppe von APIs, die einen verbesserten Zugriff auf die erweiterten Funktionen von Hochleistungshardware, z. B. 3D-Grafikbeschleunigerchips und Soundkarten, ermöglichen. Diese APIs bieten Basisfunktionen, einschließlich 2D- und 3D-Grafikbeschleunigung, Unterstützung von Eingabegeräten wie Joysticks, Tastaturen und Mäusen sowie Steuerung von Tonmischung und Tonausgabe. Die Basisfunktionen werden von den Bestandteilen unterstützt, die die DirectX Foundation-Ebene beinhaltet:

- Microsoft DirectDraw

  Die DirectDraw-API unterstützt schnelle Direktzugriffe auf die Hardwarebeschleunigerfunktionen der Grafikkarte. Sie unterstützt Standardverfahren für die Anzeige von Grafiken auf allen Karten sowie einen schnelleren, direkteren Zugriff, wenn beschleunigte Treiber verwendet werden. DirectDraw ermöglicht Programmen den geräteunabhängigen Zugriff auf Funktionen spezifischer Anzeigegeräte, ohne dass der Entwickler oder der Benutzer zusätzliche Angaben zu den jeweiligen Gerätfunktionen machen muss.

- Microsoft Direct3D Immediate Mode

  Die Direct3D Immediate Mode-API (Direct3D) dient als Schnittstelle zu den 3D-Renderingfunktionen, die in den meisten neuen Grafikkarten integriert sind. Direct3D ist eine 3D-API, die Anwendungen unabhängig von der verwendeten Hardware eine leistungsfähige Kommunikation mit der Beschleunigungshardware ermöglicht. Direct3D unterstützt Anwendungsentwickler mit zahlreichen erweiterten Funktionen wie Pufferung mit schaltbarer Tiefe (mit Z- oder W-Pufferung), Flat- und Gouraud-Shading, mehreren Lichtquellen und Beleuchtungsarten, Material- und Strukturunterstützung, Transformationen und Clipping. Bei Verfügbarkeit der entsprechenden Gerätetreiber stellt Direct3D eine umfassende Hardwarebeschleunigung sicher und beinhaltet zudem eine integrierte Unterstützung für spezielle CPU-Befehlssätze, wie z. B. Intel MMX sowie AMD 3DNow!.

- Microsoft DirectSound

  Die DirectSound-API schafft die Verbindung zwischen Programmen und den Funktionalitäten der Soundkarte. Sie ermöglicht die Aufzeichnung und Wiedergabe von Wavesound, Hardwarebeschleunigung und direkten Zugriff auf das Audiogerät.

- Microsoft DirectMusic

  Im Gegensatz zur DirectSound-API, die digitale Klangmuster aufnimmt und wiedergibt, arbeitet DirectMusic mit meldungsbasierten Musikdaten, die entweder über die Soundkarte oder den integrierten Softwaresynthesizer in digitale Audiodaten konvertiert werden. DirectMusic unterstützt die Eingabe im MIDI-Format (Musical Instrument Digital Interface) und gibt Anwendungsentwicklern auch die Möglichkeit, dynamische Soundtracks zu erstellen, die durch Benutzereingaben ausgelöst werden.

- Microsoft DirectInput

  Die DirectInput-API bietet eine erweiterte Eingabe für Spiele und verarbeitet Eingaben von Joysticks sowie anderen entsprechenden Geräten, einschließlich Maus, Tastatur, und anderen Gamecontrollern, z. B. Force Feedback-Devices.

Zusammen mit der DirectX Foundation-Ebene bietet die DirectX Media-Ebene High-Level-Dienste, durch die Animationen, Mediendatenströme (Übertragung und Anzeige von Ton- und Videodownloads aus dem Internet) und Interaktivität unterstützt werden. Wie die DirectX Foundation-Ebene besteht auch die DirectX Media-Ebene aus verschiedenen integrierten Bestandteilen:

- Microsoft Direct3D Retained Mode

  Die Direct3D Retained Mode-API bietet eine hochentwickelte Unterstützung für erweiterte 3D-Echtzeitgrafiken. Sie umfasst die integrierte Unterstützung von Grafiktechniken wie Hierarchien und Animationen. Der Direct3D Retained Mode wurde aus dem Direct3D Immediate Mode weiterentwickelt.

- Microsoft DirectAnimation

  Die DirectAnimation-API ermöglicht die Integration und Animation unterschiedlicher Medientypen, z. B. von 2D-Bildern, 3D-Objekten, Klängen, Video, Text und Vektorgrafiken.

- Microsoft DirectPlay

  Die DirectPlay-API unterstützt Spielverbindungen über ein Modem, das Internet oder ein LAN. Sie vereinfacht den Zugriff auf Kommunikationsdienste und bietet die Möglichkeit, dass Spiele unabhängig vom zugrunde liegenden Protokoll oder Onlinedienst miteinander kommunizieren können.

- Microsoft DirectShow

  Die DirectShow-API gibt Multimediadateien vom lokalen System oder von Internetservern wieder und erfasst Multimediadatenströme von Geräten wie Videoaufzeichnungskarten. Es werden dabei verschiedenste Video- und Audioformate unterstützt, einschließlich MPEG-, AVI- (Audio-Video-Interleaved) und WAV-Formaten.

- Microsoft DirectX Transform

  Die DirectX Transform-API versetzt Anwendungsentwickler in die Lage, digitale Bilder zu erstellen, zu animieren und zu bearbeiten. DirectX Transform arbeitet sowohl mit 2D- als auch mit 3D-Inhalten, die zum Erstellen eigenständiger Programme oder dynamischer Plug-Ins für Webgrafiken verwendet werden können.

Die im Vergleich zu OpenGL wichtigste API ist Direct3D, welche lange Zeit in diesem Vergleich deutlich unterlegen war. Die kontinuierliche Weiterentwicklung der API hat Direct3D jedoch im Laufe der Zeit sehr mächtig und stabil gemacht. In den Augen vieler Entwickler ist daher sogar Direct3D mittlerweile zur Norm für grafische Darstellung auf Windows-Plattformen geworden. Microsoft arbeitet hierzu sehr eng mit den Grafikkartenhersteller zusammen, um sicherzustellen, dass neue Hardwaremerkmale direkt unter Direct3D unterstützt werden. Oft unterstützt Direct3D sogar bestimmte Merkmale, bevor es die Karten selbst tun.

Die Struktur von Direct3D und DirectX im Allgemeinen unterscheidet sich deutlich von OpenGL, wobei jedoch bemerkt sei, dass sich Direct3D im Laufe der Zeit diesbezüglich mehr und mehr an OpenGL angenähert hat. DirectX basiert grundsätzlich auf dem COM Objektmodell. Die direkte Programmierung in DirectX ist relativ komplex, bereits die Erzeugung eines einfachen Dreiecks benötigt relativ viel Programm Code. Daher hat Microsoft eine Menge von Klassen, die ständig wiederkehrende Funktionalitäten implementieren, in einer Library zusammengefasst, den so genannten DirectX Common Files. Hierdurch bleibt u. a. die komplette Initialisierung vor dem Programmierer verborgen und sogar kompliziertere Aufgaben wie die Auflistung des Leistungsspektrums der Grafikkarte werden von den Common Files behandelt. Da die zugehörigen Klassen um andere DirectX-Klassen herum aufgebaut sind, können sie nur unter C++ verwenden werden.

Direct3D weist gegenüber OpenGL einige Vorteile auf. So enthält DirectX bereits seit Version 8 programmierbare Pixel- und Vertex-Shader, die es erlauben, Teile der Visualisierungspipeline durch individuellen Code zu ersetzen. Shader wurden ursprünglich in einer Grafiksprache geschrieben, die sehr stark Assemblercode ähnelt; mittlerweile existieren aber auch viele verschiedene Ansätze für High-Level Shadersprachen. Es sei bemerkt, dass auch OpenGL Pixel- und Vertex-Shader unterstützt, jedoch nur über Erweiterungen, bei denen sich die Grafikkartenhersteller noch nicht auf einen Standard einigen konnten. Ein weiterer Vorteil von Direct3D liegt in den integrierten Funktionen zur Ermittlung, welche Grafikhardware auf einem System verfügbar ist. Die Tatsache, dass DirectX COM verwendet, liefert eine erprobte Methode, um Änderung einzuführen, ohne bereits vorhandenen Code ändern zu müssen. Dies basiert auf der Regel, dass sich eine COM-Schnittstelle nicht ändern darf. Oberflächlich betrachtet erscheint dies zunächst von Nachteil, jedoch hat diese Vorgehensweise zwei entscheidende Vorteile. Zum einen kann eine neue (und beliebig verschiedene) Version einer Schnittstelle erstellt werden, obwohl sich eine gegebene Schnittstelle nicht ändern darf. Zum anderen darf eine Schnittstelle auch nicht mehr entfernt werden, so dass auch mit alten DirectX-Versionen entwickelte Programme unter neueren Versionen laufen (Die alte Schnittstelle ist ja immer noch unverändert vorhanden und kann weiter verwendet werden).

Die Schwächen von DirectX liegen beispielsweise in der Tatsache, dass es nur etwa einmal pro Jahr aktualisiert wird, was gerade im Hinblick auf die enorme Entwicklungsgeschwindigkeit bei den Grafikkarten als deutlich zu selten anzusehen ist. Die Programmierung in Direct3D ist relativ aufwendig, wodurch sich der Einstieg in DirectX schwieriger gestaltet als bei OpenGL. So benötigt allein die Anzeige eines Dreiecks unter Direct3D 7 ca. 800 Zeilen Code und in Direct3D 8 trotz starker Optimierungen immerhin immer noch ca. 200 Zeilen. Ein äußerst schwer wiegender Nachteil von DirectX ist die fehlende Portierbarkeit. DirectX ist lediglich für Windows verfügbar. Dies schließt daher die Nutzung von DirectX

von vorneherein aus, wenn man plattformunabhängige Entwicklungen betreiben möchte. Auch stellt Direct3D keinen offenen Standard dar und nur Microsoft hat die Entscheidungsgewalt, welche Funktionalitäten in neuen Releases integriert oder verändert werden.

3.1.1.2 OpenGL

OpenGL ([90],[103]-[105],[129],[131]) wurde im Jahr 1992 von Silicon Graphics als Nachfolgeversion der unter dem Namen Iris GL bekannten 3D-API vorgestellt. OpenGL wurde als ein offener Standard entwickelt, stellt aber nicht wie oftmals behauptet ein Open Source System dar. Open steht hier für „offener Standard“, wogegen GL für Grafikbibliothek (Graphics Library) steht. OpenGL wird kontinuierlich durch das Architectural Review Board (ARB) überprüft und evaluiert. Das ARB besteht dabei aus Vertretern von großen Gesellschaften, die in der Grafikindustrie engagiert sind; die wichtigsten Vertreter sind dabei 3D Labs, SGI, Apple, nVidia, ATI, Intel, id Software und Microsoft. Auch aus diesem Grund ist OpenGL im Gegensatz zu DirectX auf vielen verschiedenen Plattformen verfügbar.

Für Windows existieren zwei wichtige Implementierungen von OpenGL: eine Version von SGI und eine von Microsoft. Die Microsoft-Version basiert dabei aber auf der SGI-Implementierung. Da letztere jedoch nicht mehr aktiv unterstützt wird, wird im Allgemeinen die Verwendung der Microsoft-Version empfohlen.

Neben den beiden gerade beschriebenen Implementierungen gibt es noch eine inoffizielle OpenGL-Umsetzung: Mesa3D. Mesa3D läuft sehr stabil, ist ein Open Source Programm und ist auf vielen Plattformen verfügbar. Die Entwickler von Mesa3D besitzen jedoch keine OpenGL-Lizenz, daher darf rein rechtlich gesehen Mesa3D nicht als OpenGL-Implementierung bezeichnet werden, obwohl es natürlich dieselbe Syntax verwendet. Der einzig schwerwiegendere Nachteil im Vergleich zu den beiden kommerziellen Implementierungen ist die Tatsache, dass Mesa3D nicht immer die Features der verwendeten Hardware ausnutzen kann und deshalb oft Funktionalitäten in Software emuliert.

OpenGL wird bereits seit vielen Jahren in den verschiedensten Bereichen der Technologieindustrie verwendet. Es wurde von Anfang an mit einer klaren Zukunftsvision entwickelt, und ist dabei jederzeit stabil und konsistent geblieben. Seine klare Struktur sichert Open GL bis heute seinen großen Nutzerstamm. Andererseits hat aber die ständige Kontrolle durch den ARB zu einer relativ langsamen Entwicklung der Spezifikation des OpenGL-Kerns geführt. Neue Features, die noch nicht in den aktuellen Standard übernommen wurden, können über einen Erweiterungsmechanismus (Extensions) eingebunden werden.

Lange Zeit war dies von relativ geringer Bedeutung, da OpenGL hauptsächlich für die Verwendung mit High-End-Grafik-Workstations, welche mit professionellen Grafikkarten bestückt sind, entworfen wurde. Da aber mittlerweile Consumer-Grafikkarten die Performance, die noch vor 10 Jahren als absolutes State-of-the-Art galt, eingeholt haben bzw. vielfach sogar weit übertreffen, mussten die Entwickler aufgrund der langsamen Entwicklung des OpenGL-Kerns mehr und mehr vom Erweiterungsmechanismus Gebrauch machen, um mit neuen Hardwaremerkmalen Schritt halten zu können. Solche Erweiterungen sind zwar leistungsfähig, führen aber auch häufig zu einem sehr unübersichtlichen Code. Das ARB scheint dieses Problem jedoch erkannt zu haben und aktualisiert die Kernspezifikation in jüngster Zeit häufiger.

OpenGL ist von seinem Kern her ein Zustandsautomat (state machine), der überwacht, wie Primitive verarbeitet und gerendert werden. OpenGL verwendet ein prozedurales Modell (d. h. Funktionen), um den Zustandsautomaten zu modifizieren und um Daten zu diesem zu übermitteln. Aufgrund der hohen Effizienz dieses Mechanismus ist der für eine bestimmte Aufgabe zu entwickelnde Code normalerweise relativ kurz und in der Regel auch leicht zu verstehen, wodurch vor allem auch die Fehlerbeseitigung deutlich erleichtert wird. Obwohl OpenGL selbst prozedural ist, kann es vom Programmierer in linearer, modularer oder OOP-Weise verwendet werden. OpenGL verbirgt dabei die meisten Details über die darunter liegende Hardware vor dem Entwickler.

Alle OpenGL-Funktionen basieren auf einer vorgegebenen Namenskonvention: *glFunctionName [argtypes]*. Als Beispiel für eine entsprechend definierte OpenGL-Funktion sei an dieser Stelle *glVertex3f* genannt. Das Prefix *gl* weißt zunächst darauf hin, dass es sich bei diesem Aufruf um eine OpenGL-Funktion handelt. *Vertex* besagt, dass das Argument ein Eckpunkte ist, *3*, dass dieser Punkt dreidimensional angegeben wird, und *f*, dass das Argument ein Float-Wert ist. Zur Unterstützung der Portabilität von OpenGL-Programmen sind in OpenGL spezielle Typen deklariert. Diese beginnen mit dem Prefix *GL*, evtl. gefolgt von einem *u* bei vorzeichenlosen (unsigned) Werten. Danach wird der eigentliche Typname angegeben, z. B. *short*, *long*, oder *float*. Als Beispiel sei hier der Integer-Typ *GLuint* genannt. Das eigentliche Rendering erfolgt in OpenGL zwischen den beiden Aufrufen *glBegin* und *glEnd*. Insgesamt gesehen ist die OpenGL-Syntax gerade bei Verwendung der Programmiersprache C deutlich intuitiver und einfacher nutzbar als die Direct3D-Syntax.

Zusammenfassend lässt sich feststellen, dass OpenGL viele Stärken besitzt. Zum einen ist es äußerst portabel und läuft daher auf beinahe jeder existierenden Plattform. Im Hinblick auf DirectX ist dies ein großer Vorteil, selbst auf der Microsoft-Plattform Windows NT 4.0, auf der Microsoft selbst nur eine fünf Versionen alte Implementierung von DirectX anbietet, läuft die aktuelle OpenGL-Release stabil.

Der Grund hierfür liegt in der Open Standard Struktur von OpenGL. Jede Firma, die mit ihrer Plattform OpenGL unterstützen möchte, kann eine Lizenz von SGI kaufen und dann die API für diese Plattform implementieren.

OpenGL bietet sowohl in seinem Kern als auch durch Erweiterungen ein breites Spektrum von Features und wird daher in den unterschiedlichsten Anwendungen verwendet. Die integrierte Erweiterungsmöglichkeit erlaubt es OpenGL, auch neueste Hardwarefeatures zeitnah zu nutzen. Neben der Anzahl der gebotenen Features haben vor allem die hohe Stabilität und die Unterstützung einer Vielzahl von Plattformen zum Erfolg von OpenGL beigetragen. Im Vergleich zu DirectX wird OpenGL in der Industrie (3D-Modeling, Bildaufbereitung, CAD usw.) und auch bei militärischen Anwendungen verstärkt eingesetzt, während sich die Spieleindustrie bis auf wenige Ausnahmen auf Direct3D konzentriert.

Obwohl die beschriebene Erweiterungsmöglichkeit von OpenGL sehr mächtig ist, bringt diese auch zahlreiche Nachteile mit sich. So ist der entstehende Programmcode in der Regel sehr unübersichtlich und wird nicht von allen Compilern korrekt unterstützt. Zudem sind viele, vor allem neuere, Erweiterungen nur für spezielle Grafikkarten bzw. -hersteller entwickelt worden und untergraben somit die eigentliche Plattformunabhängigkeit des Kerns.

3.1.1.3 Java 3D

Die Java 3D-Klassenbibliothek ([108],[114]-[116]) ist eine Schnittstelle für die Darstellung und Manipulation dreidimensionaler Szenerien innerhalb von Java-Applikationen und -Applets und stellt ein Add-On zu Sun's Java Virtual Machine dar. Die Java 3D-API wurde von den Firmen Sun, SGI, Apple und Intel entwickelt und erstmals 1997 auf der SIGGRAPH vorgestellt. Java 3D implementiert Objekte und Methoden zur Repräsentation einer Szene und verwendet für deren Darstellung eine lokal vorhandene Rendering-API (OpenGL, unter Windows auch DirectX). Dadurch kann mittels Java 3D ebenfalls Nutzen aus der verfügbaren 3D-Beschleunigungshardware beim Rendern von Szenen gezogen werden.

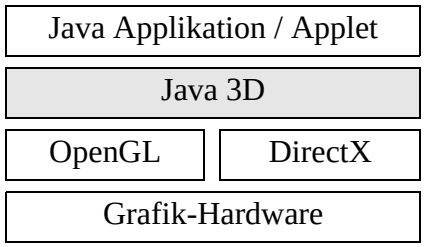

Abbildung 3-1: Positionierung von Java 3D.

Der Zugriff auf solch lokale und plattformabhängige Renderingschnittstellen macht für jede Plattform eine spezielle Java 3D-Version nötig. Zurzeit ist Java 3D als Release 1.3.1 für Windows (als Varianten für OpenGL und DirectX) und Solaris SPARC erhältlich. Die Schnittstelle zu Applikationen oder Applets, welche Java 3D-Funktionalitäten verwenden, ändert sich jedoch nicht – eine installierte Java 3D-Bibliothek vorausgesetzt, läuft das Programm somit auf allen Plattformen.

In Java 3D wird eine dreidimensionale Szenerie als virtuelles Universum beschrieben, in welchem Java 3D-Objekte als Knoten in einer Baumstruktur, dem so genannten Szenengraphen, angeordnet sind. Die einzelnen Verbindungen zwischen den Knoten stellen immer gerichtete Beziehungen dar, Szenengraphen dürfen nach ihrer Definition keine Zyklen enthalten. Der Java 3D-Szenengraph stellt daher einen gerichteten antizyklischen Graphen (DAG – directed acyclic graph) dar. Durch die hierdurch vorgegebene Struktur der Szene kann diese deutlich komfortabler gehandhabt werden als z. B. direkt in OpenGL oder DirectX. Java 3D nimmt dem Benutzer hierzu Aufgaben ab, die bei direkter Benutzung durch die Anwendung implementiert werden müssten. Ein Szenengraph organisiert und kontrolliert das konsistente Rendering seiner Objekte. Für die Darstellung wird der Szenengraph entsprechend ausgewertet und wenn möglich in eine auf Renderingperformance optimierte Darstellung konvertiert. Diese Konvertierung ist von der zugrunde liegenden Rendering-API abhängig, im Falle von OpenGL kann dies z. B. die Erzeugung einer Renderlist sein. Der Renderer kann dabei ein Objekt unabhängig von anderen Objekten rendern, da in Java 3D verschiedene Zweige eines Baumes keine gemeinsamen Zustände besitzen können. Bei Änderungen an der Szene (Geometrie, Transformationen oder Struktur des Graphen) erfolgt jeweils eine erneute Auswertung.

Die Hierarchie des Szenengraphen erlaubt eine natürliche räumliche Gruppierung der sich auf Blattebene des Baumes befindlichen geometrischen Objekte. Die Aufgabe der inneren Knoten liegt in der Gruppierung ihrer Kinder. Ein Gruppenknoten definiert auch eine räumliche Schranke, welche die durch seine Nachkommen definierte Geometrie enthält. Diese räumliche Gruppierung ermöglicht eine effiziente Implementierung von Operationen wie Abstandsbestimmung, Kollisionserkennung, View Frustum Culling (Ausblenden von Objekten außerhalb des Sichtbereiches) und Occlusion Culling (Ausblenden von verdeckten Szenenteilen).

Jeder Szenengraph teilt sich in einen Content- und Viewing-Untergraphen. Während der Viewing-Teilbaum nur aus wenigen Knoten besteht, umfasst der Content-Teilbaum oft mehrere 1.000 Knoten. Die Wurzel eines jeden Szenengraphen besteht aus den beiden Superstrukturkomponenten *VirtualUniverse* und *Locale*. Normalerweise benötigt eine Applikation nur ein *VirtualUniverse*, es können jedoch auch mehrere hiervon definiert sein. Ein *VirtualUniverse*-Objekt besteht

aus einer Liste von *Locale*-Objekten, welche als Container für eine Menge von Teilgraphen im Szenengraph, die von einem *BranchGroup*-Knoten ausgehen, dienen. Weiterhin definiert ein *Locale*-Objekt auch seine Position innerhalb des virtuellen Universums mit Hilfe von hochauflösenden (*HiRes*) Koordinaten. Diese Koordinaten dienen als Ursprung für alle innerhalb des *Locale*-Objektes enthaltenen Szenengraphobjekte (vergleiche Abbildung 3-2).

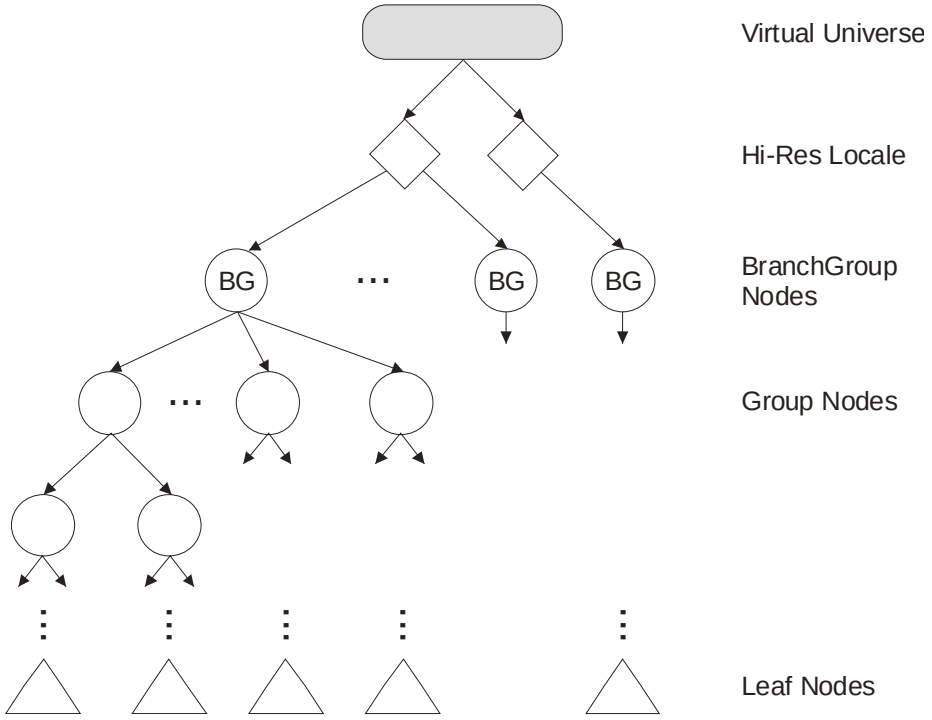

Abbildung 3-2: Prinzipieller Aufbau des Java 3D Szenengraphen.

TransformGroup-Knoten bestimmen die relativ zu dem durch das *Locale*-Objekt vorgegebenen Ursprung die Position, die Skalierung und die Orientierung im Raum aller geometrischen Objekte des zugehörigen Teilbaumes. Die eigentlich darzustellende Information wird in den Blättern des Baumes gehalten, z. B. die Geometrie, Lichtquellen oder Soundobjekte. In Java 3D ist hierzu die Klasse *leaf node* definiert, eine abstrakte Klasse für alle Szenengraphknoten, die keine Kinder haben. Der Blattknoten *Shape3D* dient zur Angabe der geometrischen Objekte und enthält eine Liste von einem oder mehreren *Geometry*-Objekten und ein einzelnes *Appearance*-Objekt. Während die *Geometry*-Objekte die geometrischen Daten des Shape-Knotens definieren, beschreibt das *Appearance*-Objekt das Erscheinungsbild dieses Objektes durch Attribute wie Farbe, Material, Textur usw. Der *Behavior*-Knoten erlaubt einer Anwendung, die Manipulation des Szenengraphen zur Laufzeit zu manipulieren und ermöglicht somit zum Beispiel die Animation von Objekten, die Verarbeitung von Tastatur- und Mauseingaben sowie Picking. Hierzu existieren mehrere vordefinierte Verhalten sowie die Möglichkeit, benut-

zerspezifischen Java-Code als *Behavior*-Objekt zu integrieren. Dieser Code manipuliert dann während der Laufzeit die Transformationsmatrix, die über den *TransformGroup*-Knoten mit den zugehörigen geometrischen Objekten verknüpft ist.

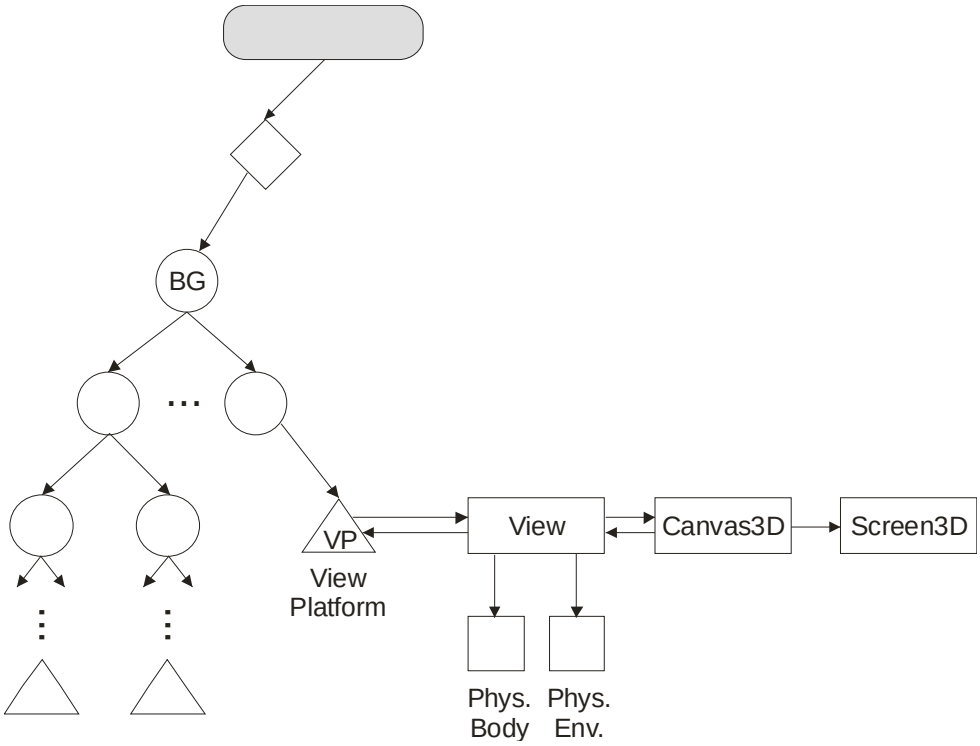

Abbildung 3-3: Viewing-Objekte im Szenengraph.

Wie bereits erwähnt, existiert neben dem Content- auch ein Viewing-Untergraph, in welchem die Sichtparameter definiert werden. Java 3D spezifiziert hierzu unterhalb des *ViewPlatform*-Blattes zur Definition der Ansicht einer Szene fünf Viewing-Objekte: *View, Canvas3D, Screen3D, PhysicalBody* und *PhysicalEnvironment*. Das *View*-Objekt stellt dabei das zentrale Java 3D-Objekt für die Beschreibung der Ansicht einer Szene dar. Alle Sichtparameter in Java 3D sind entweder direkt im *View*-Objekt oder in den mit diesem verbundenen Objekten enthalten. Die Java 3D-API unterstützt mehrere, sogar gleichzeitig aktive *View*-Objekte, von denen jedes in ein oder mehrere Zeichenflächen (Canvas) rendern kann. Das *Canvas3D*-Objekt kapselt alle Parameter des Canvas, in die gerendert werden soll. Wenn ein *Canvas3D*-Objekt einem *View*-Objekt angehängt wird, wird die Ausgabe des Java 3D-Renderers auf die zugehörige Zeichenfläche umgeleitet. Dabei können mehrere *Canvas3D*-Objekte dem gleichen *View*-Objekt zugeordnet sein. Das *Screen3D*-Objekt beinhaltet alle Parameter des physischen Bildschirms, in welchem der Canvas liegt (wie z. B. die physikalische Auflösung des Bildschirms in Pixeln). Das *PhysicalBody*-Objekt dient zur Angabe von Körpercharakteristiken des Endbenutzers, wie z. B. Kopfposition, rechte und linke Augenposition,

Augenabstand usw. Das *PhysicalEnvironment*-Objekt beinhaltet alle mit der physikalischen Umgebung verbundenen Parameter, wie zum Beispiel die für Tracker nötigen Kalibrierungsinformationen. Für Standardanwendungen, die kein Head- bzw. Eyetracking unterstützen, werden von Java 3D Standardwerte für *PhysicalBody* und *PhysicalEnvironment* definiert. Der Viewing-Untergraph ist in Abbildung 3-3 skizziert.

Die Knotenobjekte des Szenengraphen befinden sich dabei in einem von drei möglichen Zuständen: Direkt nach der Erzeugung, d. h. vor dem Einfügen in den Graphen, ist ein Knoten *detached*. In diesem Zustand können alle Attribute ausgelesen und verändert werden. Durch das Einfügen in den Szenengraphen ändert der Knoten seinen Zustand von *detached* in *live*. In diesem Zustand können nun nur noch Operationen, die vorher mittels `setCapability()` freigegeben wurden, angewendet werden. Sobald der Teilgraph, zu dem ein Knoten gehört, mittels `compile()` optimiert wird, wechselt der Zustand zu *compiled*. Durch die so durchgeführte Optimierung auf die zugrunde liegende Rendering-API sind Änderungen nicht mehr oder nur noch sehr eingeschränkt möglich, beispielsweise kann Geometrieinformation in diesem Zustand nicht mehr verändert werden.

3.1.1.4 VTK

Das Visualisierungstoolkit VTK (Visualization ToolKit; [71],[98]-[100]) ist eine frei verfügbare C++ - Klassenbibliothek zur grafischen Datenverarbeitung und Visualisierung. VTK stellt ein Open Source Produkt dar, welches sich seit etwa 10 Jahren am Markt befindet. Der Hersteller, Kitware Inc., leitet eine große Entwicklergemeinde, die im Laufe der Zeit eine sehr große Zahl an Komponenten erstellt hat.

Die Entwickler von VTK wollten vermeiden, ein großes, monolithisches System aufzubauen, das keine gezielte Entwicklung von flexibler Software erlaubt. Daher ist VTK als objektorientierte Klassenbibliothek zur Entwicklung datenflussorientierter Visualisierungsanwendungen konzeptioniert. Toolkits wie VTK ermöglichen im Allgemeinen den Aufbau komplexer Anwendungen aus kleinen Teilstücken. Hierzu müssen alle Einzelteile mit exakt definierten Schnittstellen versehen werden, um diese dann in größere Systeme integrieren zu können. Neben der Objektorientierung versuchten die Entwickler, die Vorteile von kompilierten und interpretierten Sprachen geschickt zu mischen. Kompilierte Sprachen erlauben in der Regel eine deutlich höhere Performance als interpretierte Sprachen, dafür bieten interpretierte Sprachen eine größere Flexibilität. Daher ist der Kern von VTK unter Verwendung einer kompilierten Sprache programmiert, während High-Level-Anwendungen mit Hilfe einer interpretierten Sprache umgesetzt werden können (siehe Abbildung 3-4).

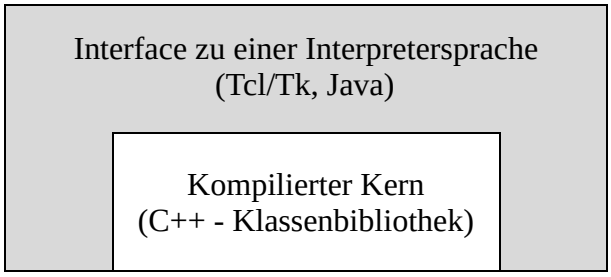

Abbildung 3-4: Architektur von VTK.

Vom Systemdesign her gesehen lassen sich bei VTK zwei verschiedene Ansätze erkennen: zum einen das grafische Darstellungsmodell, welches ein abstraktes Modell für 3D-Grafik darstellt, und zum anderen das Visualisierungsmodell, welches ein Datenflussmodell des Visualisierungsprozesses abbildet. Das grafische Darstellungsmodell fasst die essentiellen Features eines 3D-Grafiksystems zusammen und besteht bei VTK aus neun verschiedenen Objekten:

- Render Master (Koordination der geräteunabhängigen Methoden und Erzeugung der Render Windows),
- Render Window (Fenstermanagement),
- Renderer (Koordination des Renderings von Lights, Cameras und Actors),
- Light (Beleuchtung der Szene),
- Camera (Definition der Kamera),
- Actor (Objekt, welches vom Renderer gezeichnet wird und durch Mapper-, Property- und Transform-Objekte definiert ist),
- Property (Rendering-Attribute),
- Mapper (geometrische Definition des Actors) und
- Transform (4x4-Transformationsmatrix mit Methoden zur Modifikation der Matrix).

Teilweise besitzen diese Objekte direkte geräteabhängige Gegenstücke. Werden entsprechende generische Objekte in VTK erzeugt, so werden automatisch und transparent für den Benutzer auch die hardwareabhängigen Objekte erzeugt, die ihrerseits auf die verwendete Grafikbibliothek abgebildet werden.

Das VTK-Visualisierungsmodell basiert auf dem auch von vielen kommerziellen Systemen verwendeten Datenflussparadigma. Nach diesem Paradigma werden die einzelnen Module durch ein Netzwerk verbunden und die Module führen ihre jeweiligen algorithmischen Operationen auf den Daten aus, die aktuell durch dieses Netz fließen. Die Ausführung dieses Visualisierungsnetzwerkes wird entweder

durch Datenanforderung (bedarfsgesteuert) oder durch Benutzerinteraktionen (ereignisgesteuert) gesteuert. Der Vorteil dieses Modells ist, dass es sehr flexibel ist und schnell an verschiedene Datentypen oder neue Algorithmen angepasst werden kann.

Das VTK-Visualisierungsmodell besteht aus zwei Basisobjekttypen: Prozess- und Datenobjekte. Dabei symbolisieren die Prozessobjekte die Module bzw. algorithmischen Teile des Visualisierungsnetzwerkes, während die Datenobjekte Zugriffsoperationen auf die Daten, die durch das Netzwerk fließen, repräsentieren. Abbildung 3-5 zeigt das VTK-Visualisierungsmodell am Beispiel von drei Prozessobjekten A, B und C, welche als Ein- und / oder Ausgabe ein oder mehrere Datenobjekte haben. Im Visualisierungsmodell stellt A ein SourceObject, B ein FilterObject und C ein MapperObject (auch SinkObject genannt) dar. Die Pfeile zwischen den einzelnen Objekten markieren dabei den zugehörigen Datenfluss.

Die SourceObjects stellen den Beginn des Datenflusses dar und generieren durch Einlesen entsprechender Dateien bzw. prozedural die benötigten Datensätze. Sie besitzen hierfür Methoden zur Initialisierung des Datenflusses und stellen den im Netzwerk nachgestellten Knoten (z. B. FilterObjects) die Daten zur Weiterverarbeitung zur Verfügung. Ein Beispiel für ein SourceObject in VTK ist die Klasse *vtkPolyDataReader*, welche polygonale Daten (ASCII- oder Binärformat) im VTK-Format einlesen kann. FilterObjects verarbeiten die erhaltenen Daten und reichen sie dann an nachgeschaltete Filter- oder MapperObjects weiter. Ein Beispiel für ein FilterObject ist die Klasse *vtkMarchingCubes*, welche als Eingabe ein Volumen (z. B. eine dreidimensionale strukturierte Punktmenge) nimmt und hieraus eine oder mehrere Isoflächen generiert. Am Ende des Datenflusses stehen dann die MapperObjects, welche die erhaltenen Daten auf grafische Primitive (Punkte, Linien, Dreiecke) abbilden und diese anschließend darstellen. Ein Beispiel stellt die Klasse *vtkPolyDataMapper* dar, welche polygonale Datensätze (*vtkPolyData*) auf Grafikprimitive abbildet. Unterstützt werden die MapperObjects durch eine Reihe von Klassen, die für das Rendering zuständig sind (z. B. *vtkRenderWindow, vtkLight, vtkCamera*).

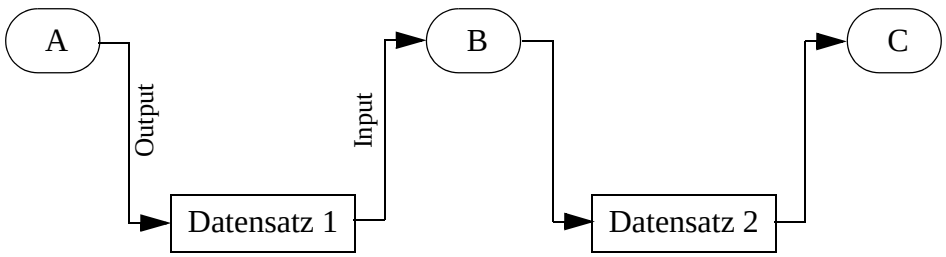

Abbildung 3-5: VTK-Visualisierungsmodell.

Im Folgenden soll kurz auf die in VTK zur Verfügung stehenden Datenstrukturen und Datenobjekte eingegangen werden. Ziel der Visualisierung ist es, Daten, die z. B. durch Simulationen, Laserscanner oder durch Berechnungen generiert wurden, aufzubereiten und grafisch darzustellen. Die Daten sind dabei diskret, d. h. man hat eine endliche Menge von Punkten, an denen bestimmte Attribute (z. B. Temperatur, Luftdruck, Geschwindigkeit usw.) gemessen oder berechnet wurden. Die Messpunkte können hierbei strukturiert oder unstrukturiert verteilt sein, d. h. sie sind also z. B. in einem regelmäßigen Gitter angeordnet oder befinden sich an beliebigen Raumpositionen. Strukturierte Daten sind für eine effiziente Speicherung sowie einen schnellen Datenzugriff geeignet, während unstrukturierte Daten adaptiv so angeordnet werden können, dass sie eine möglichst gute Approximation der zu visualisierenden Informationen ermöglichen.

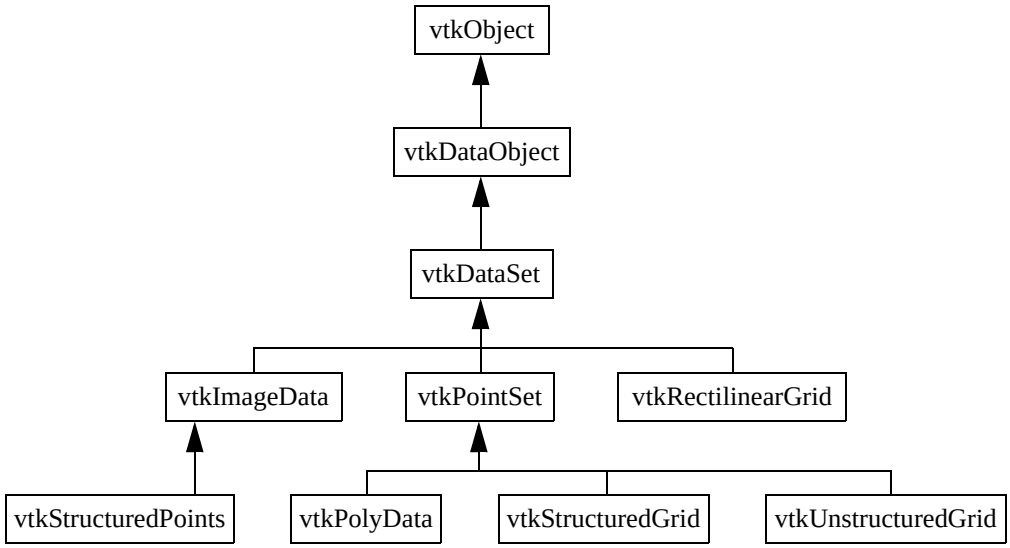

Abbildung 3-6: VTK-Datensatztypen: Klassenhierarchie.

VTK unterscheidet zwischen der Topologie (realisiert die Organisation und den Zugriff auf Messpunkte) und der Geometrie (Koordinaten der Messpunkte) eines Datensatzes. Die eigentlichen Messwerte (Skalare, Vektoren, Normalen, Texturkoordinaten und Tensoren) können dann dem ganzen Datensatz, der Topologie oder der Geometrie zugeordnet werden. Da strukturierte Daten auf vielerlei Arten angeordnet sein können, wird in VTK die Topologie eines Datensatzes durch eine Reihe verschiedener topologischer Bausteine, den Zellen (Cells), beschrieben. Beispiele für solche Cells sind Vertex, Polyvertex, Line, Polyline, Triangle und Triange strip. Aufbauend auf diesen topologischen Elementen sind die Datensätze vtkPolyData (polygonale Daten mit unstrukturierter Topologie und Geometrie), vtkUnstructuredPoints (unstrukturierte Punktmenge, keine Topologie), vtkStructuredPoints (in Bezug auf Topologie und Geometrie strukturierte Punktmenge, angeordnet auf einem regulären, rechteckigen Gitter), vtkStructuredGrid (reguläre

Topologie, irreguläre Geometrie) und vtkRectilinearGrid (reguläre Topologie, semi-reguläre Geometrie) als Klassen realisiert. Abbildung 3-6 zeigt die Klassenhierarchie der gerade vorgestellten VTK-Datensatztypen.

Das VTK-Toolkit ist sowohl für Unix als auch für Windows-basierte Systeme verfügbar. Es ist klar erkennbar von Entwicklern für Entwickler geschrieben, die Arbeiten mit VTK gestalten sich aus diesem Grund auch nicht sehr benutzerfreundlich. Die von Kitware angebotenen Online-Dokumentationen und -Hilfen sind eher knapp gehalten und wenden sich fast ausschließlich an erfahrene Nutzer. Für Einsteiger sind die Erklärungen häufig zu knapp gehalten, um in einer vernünftigen Zeit zu einem brauchbaren Ergebnis zu kommen. Ausführlichere und weiterführende Kurse und Bücher zu VTK werden zwar ebenfalls angeboten, aber nur kommerziell vertrieben. Da das System an sich aber kostenlos ist, hat sich im Laufe der Zeit eine größere Usergemeinschaft entwickelt, die vor allem die Vielzahl der enthaltenen Komponenten schätzt. Bedingt durch den Open Source Ansatz arbeitet kontinuierlich eine Vielzahl von voneinander unabhängigen Programmierern an der Überarbeitung und Ergänzung existierender Module sowie an der Erzeugung neuer Module. Problematisch wirkt sich hier aus, dass häufig die nötige zentrale Koordinierung und Kontrolle durch den Hersteller augenscheinlich nicht richtig funktioniert und entsprechend Inkompatibilitäten im Gesamtsystem entstehen sowie nicht korrekt arbeitende oder instabile Klassen in offiziellen Releases integriert sind. Auch ist der Abstand zwischen den offiziellen Releases relativ groß, so dass die Entwickler häufig auf die Nightly Builds zurückgreifen müssen, um aktuellere oder fehlerbereinigte Klassen und neue Methoden nutzen zu können. Die Nightly Builds sind jedoch kaum getestet und teilweise sind Versionen nicht korrekt installier- oder nutzbar. Auch wird auf Abwärtskompatibilität wenig Rücksicht genommen, so dass mit alten VTK-Releases entwickelte Applikationen oft nur mit hohem Aufwand angepasst oder komplett neu geschrieben werden müssen. In Firmen hat sich VTK vor allem aus diesen Gründen nicht durchsetzen können.

3.1.1.5 Zusammenfassung und Bewertung

Wie bereits zu Beginn des Abschnittes 3.1.1 erwähnt, können die betrachteten Bibliotheken in die Kategorien Low-Level-API und High-Level-API eingeteilt werden.

Die beiden den Low-Level-APIs zuzuordnenden Produkte OpenGL und DirectX sind aufgrund Ihrer Hardwarenähe äußerst performant, bedingen jedoch konzeptbedingt eine intensive Einarbeitungszeit und erfordern einen umfangreichen Implementierungsaufwand, selbst für einfache Aufgaben. Für OpenGL sprechen vor allem dessen Verfügbarkeit auf verschiedenen Plattformen, der im Vergleich zu DirectX geringere Programmieraufwand, die hohe Stabilität und die große Ver-

breitung in Industrie und Forschung. Die Vorteile von DirectX liegen dagegen in der Unterstützung modernster Hardwarearchitekturen und der damit verbundenen sehr hohen Performanz. Trotz dem höheren Programmieraufwand und der alleinigen Verfügbarkeit unter Windows wird DirectX daher in der Spieleindustrie OpenGL vorgezogen.

Die High-Level-APIs VTK und Java 3D bieten den Vorteil, dass viele Visualisierungsalgorithmen hier bereits vorhanden sind und durch einen einfachen Aufruf der zugehörigen Funktionen verwendet werden können. Die Erstellung einer entsprechenden Applikation ist somit deutlich einfacher und schneller als direkt unter einer Low-Level-API. Nachteilig wirkt sich teilweise die aufgrund des entstehenden Overheads stark gesunkene Performanz aus.

Die Vorteile von VTK liegen vor allem in der durch das OpenSource-Konzept gegebenen Möglichkeit der Einsicht und eigenen Modifikation des Source Codes, im Entfall von Lizenzkosten und Einsatzbeschränkungen und im sehr großen Funktionsumfang. Gleichzeitig führt das Open Source-Konzept auch zu einer Reihe von Nachteilen, bei VTK zudem begünstigt durch den unzureichenden Support für Einsteiger. So ist die Entwicklung des VTK-Toolkits relativ unkontrolliert und bietet an vielen Stellen kaum Abwärtskompatibilität. Auch sind selbst in offiziellen Releases teilweise große Bugs und damit verbundene Instabilitäten der generierten Applikationen zu verzeichnen, vor allem hervorgerufen durch eine fehlende bzw. unzureichende Evaluierung. Eingesetzt wird VTK (auch aus Kostengründen) vor allem in der Wissenschaft.

Java 3D zeichnet sich vor allem durch seine Szenengraph-Baumstruktur und die damit verbundene Objektorientierung, Erweiterbarkeit und Realitätsnähe aus. Weitere, äußerst wichtige Pluspunkte sind die kostenlose Verfügbarkeit, die Plattformunabhängigkeit und die flexiblen Schnittstellen für I/O-Geräte. Die Programmierung gestaltet sich verhältnismäßig einfach und durch den großen Funktionsumfang lassen sich die verschiedensten Applikationen damit entwickeln. Für eine performante Programmausführung setzt Java 3D jedoch relativ hohe Hardwareanforderungen. Nachteilig wirkt sich auch aus, dass Java 3D nicht im Sun Standard-Java-Paket J2SE enthalten ist und als „Optional Package“ entsprechend eine zusätzliche Installation notwendig macht. Wie bei VTK ist auch bei Java 3D eine oft fehlende Abwärtskompatibilität und eine im Vergleich zu direkter OpenGL- oder DirectX-Programmierung prinzipbedingt eingeschränkte Performanz zu bemerken. Zurzeit zeichnet sich die Tendenz ab, dass Java 3D keine Weiterentwicklung mehr erfährt und stattdessen ein Schwenk auf Java-OpenGL (deutlich geringerer Installationsaufwand, deutlich performanter, Szenengraph durch Xith3D-Erweiterung) erfolgt.

3.1.2 Visualisierungssysteme

Die im Rahmen dieser Arbeit betrachteten Visualisierungssysteme lassen sich zunächst grob in drei Kategorien einteilen: kommerzielle Universalsysteme, Open Source Produkte und Systeme, die rein für spezielle Einsatzzwecke zugeschnitten sind.

Die Gruppe der kommerziellen, universell einsetzbaren Produkte zeichnet sich vor allem dadurch aus, dass die Systeme von professionellen Entwicklerteams ständig auf dem neuesten Stand der Technik gehalten werden. Diese Produkte nehmen für sich in Anspruch, für ein großes Spektrum an Aufgaben geeignet zu sein. Die Benutzer (meist in Großbetrieben) müssen sich lediglich in die Benutzeroberflächen einarbeiten, welche aus diesem Grund auch in der Regel übersichtlich gehalten werden. Dafür sind diese Visualisierungssysteme allerdings auch in der Hochpreisregion angesiedelt, Lizenzpreise von 5.000 € bis 30.000 € sind keine Seltenheit, und ein Wartungsvertrag kostet – meist jährlich – etwa 50 % des Anschaffungspreises. Vertreter aus dieser Kategorie sind z. B. *AVS*, *Iris Explorer* und *Khoros*.

Die zweite auf dem Markt befindliche Gruppe sind die Open Source Systeme, welche die low- oder no-cost Alternative darstellen. Auch diese eignen sich für viele Aufgabengebiete, stellen allerdings wesentlich höhere Ansprüche an die Fertigkeiten der User. Es werden im Normalfall nur Auskunftsservices (über Email) oder Infoseiten im Netz, in einigen Fällen auch – dann aber kostenpflichtige – Trainingskurse angeboten. Die zur Verfügung stehenden Verfahren / Module erfahren nicht so schnell wie bei kommerziellen Produkten Updates; insbesondere Verfahren, die nur für spezielle Anforderungen wichtig sind, müssen häufig vom User selbst entwickelt bzw. angepasst werden. Allerdings ist meist ein großes Angebot an Lösungen für gängige Probleme vorhanden, häufig auch mehrfach, da mehrere Entwickler unabhängig voneinander dasselbe Problem angegangen sind. Häufig ändern sich auch zwischen den Releases die Schnittstellen, was aufwendige Umstellungen beim Benutzer zur Folge hat. Die Bedienung dieser Produkte ist ebenfalls eher für etwas versiertere User ausgelegt, Gelegenheitsbenutzer oder Anfänger sind großteils überfordert. Zu dieser Kategorie gehören beispielsweise *OpenDX* oder *VTK* (*VTK* wurde im Rahmen dieser Arbeit jedoch unter dem Punkt Visualisierungsbibliothek eingeordnet).

Die dritte Kategorie umfasst schließlich (ebenfalls kommerziell vertriebene) Systeme, die nur für ein ganz bestimmtes Aufgabengebiet entwickelt wurden, meist für industrielle Anforderungen und Fragestellungen. Diese Systeme sind daher optimal auf die Probleme entsprechender Benutzer zugeschnitten und bieten neben angepasster Problemlösung auch eine leicht verständliche Oberfläche und zugehörigen Support. Preislich liegen diese Systeme im mittleren Bereich (zwi-

schen 500 € und 5.000 €). Die Konkurrenz zu anderen Produkten stellt hier selten ein Problem dar, da die Entwickler häufig erst auf Auftrag ein solches System erstellen und liefern. Die Systeme sind konzeptionell bedingt nur schwer an neue Aufgaben anzupassen, haben aber in ihrem Gebiet Vorteile vor Allroundsystemen – und sei es nur der geringere Overhead an von der jeweiligen Usergruppe nicht benötigten Komponenten und dem damit einhergehenden Geschwindigkeits- und Usability-Vorteil. Zu dieser Kategorie gehört beispielsweise *inVisu PMS*.

Einige der Systeme werden in den folgenden Abschnitten näher beschrieben.

3.1.2.1 AVS

Das von Advanced Visual Systems entwickelte Visualisierungssystem *AVS* ([1]-[4]) ist ein kommerzielles Hochpreisprodukt, welches seit etwa 14 Jahren am Markt ist. Durch die lange Verfügbarkeit hat sich *AVS* als einer der Marktführer etablieren können, der mit einem stabilen Produkt, gutem Service sowie langen Lizenzlaufzeiten viele namhafte Firmen als Dauerkunden besitzt. Das Programm ist sehr vielseitig verwendbar und wird zudem auch mit maßgeschneiderten Modulen an den Kunden ausgeliefert. Die Kosten liegen bei etwa 30.000 € für das Programmpaket und ca. 5.000 € pro Jahr für den Support. Daneben gibt es noch Studentenversionen, welche stark im Preis reduziert sind (Kosten: 100 € bis 1.000 €). Allerdings umfassen diese Versionen auch nicht das komplette *AVS*-Paket, sondern stellen mehr oder weniger „abgespeckte“ Versionen dar.

AVS verwendet und unterstützt direkt etwa zehn der gängigsten Bild- und Videoformate, Einlesemodule für andere Formate sind aber ebenfalls vorhanden. Abgesehen davon verwendet *AVS* ein eigenes Datenformat für Grafiken (AVS image format).

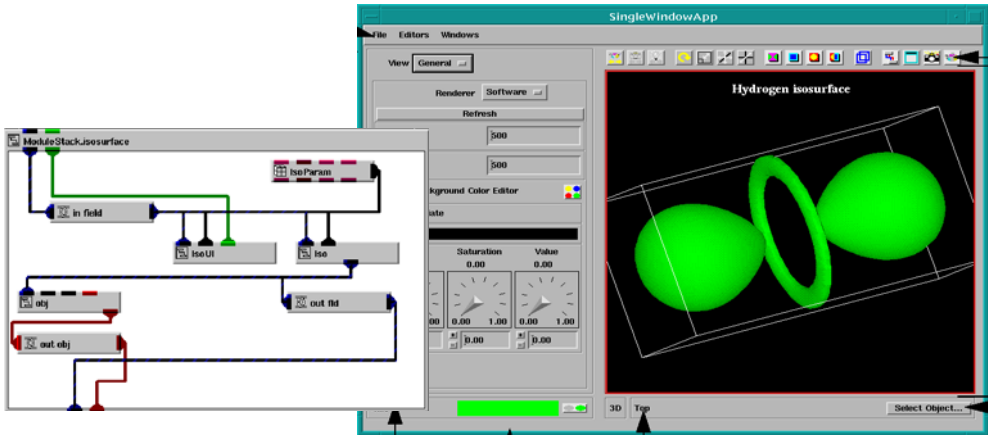

Abbildung 3-7: *AVS* DataViewer Interface.

Abbildung 3-7 zeigt die zentralen Komponenten des *AVS* DataViewer Interfaces. Das Viewer Window dient zur Darstellung und zur Interaktion mit der Visualisierung. Mittels des Editor-Fensters können zum Beispiel die Module und Datenflussnetzwerke kontrolliert werden, die zur Anwendung hinzugefügt wurden.

Das System ist auf Unix und Windows gleichermaßen verwendbar. Da durch die hauptsächliche Verwendung des Programms in Industrie und Großbetrieben von langen Vertragslaufzeiten ausgegangen werden kann, und es wohl auch bequemer war, den alten Kern des Programms zu erweitern statt einen neuen Kern zu entwickeln, ist die Benutzerfreundlichkeit für Neueinsteiger nicht so hoch wie man vermuten sollte. Da Advanced Visual Systems aber für Kunden im Rahmen der Supportverträge auch Trainingskurse anbietet und die User ja langfristig mit dem Programm arbeiten, stellt dies für *AVS* kein großes Problem dar, sondern im Gegenteil sogar einen kundenerhaltenden Vorteil – die wenigsten Firmen sind bereit, ihre Mitarbeiter auf einem ähnlich komplexen Konkurrenzsystem nochmals schulen zu lassen, wenn die Funktionsweise des anderen Systems erst einmal verstanden wurde.

AVS schlägt sogar eine Brücke von der kommerziellen zur Open Source Welt: es existiert eine relativ große Gemeinschaft von freien Entwicklern, die Module für *AVS* entwickeln und dem Herstellern zur Verfügung stellen. Je nach Bedarf werden diese Komponenten dann entweder dem Gesamtpaket hinzugefügt oder einfach zum zusätzlichen Download bereit gestellt. Diese Entwicklergemeinschaft erhält dabei kostenlosen Support vom Hersteller.

3.1.2.2 Khoros

Das von Khoral vertriebene Visualisierungssystem *Khoros* ([68],[69]) entstammt ursprünglich einem Open Source Projekt, das vor etwa sechs Jahren kommerzialisiert wurde. Das Programm legt viel Wert auf Benutzerfreundlichkeit und auf die grafische Oberfläche, wohl auch um Firmen den Umstieg von etablierten Systemen wie *AVS* oder *IRIS Explorer* zu erleichtern. Aus dem gleichen Grund bietet *Khoros* auch die Möglichkeit mit den Datenformaten von *AVS* und *IRIS Explorer* direkt zu arbeiten, ohne dass Einlese- oder Konvertierungsmodule benötigt werden.

Es existiert noch immer eine große Zahl von freien Entwicklern aus der Open Source Zeit, die immer neue Module und Komponenten für *Khoros* entwickeln. Diese Entwicklergemeinde erhält im Netz kostenlose Unterstützung durch entsprechende Daten und Tipps, welche allerdings für Einsteiger nicht geeignet sind. Als Anfänger ist man daher auf die von Khoral angebotenen Kurse angewiesen.

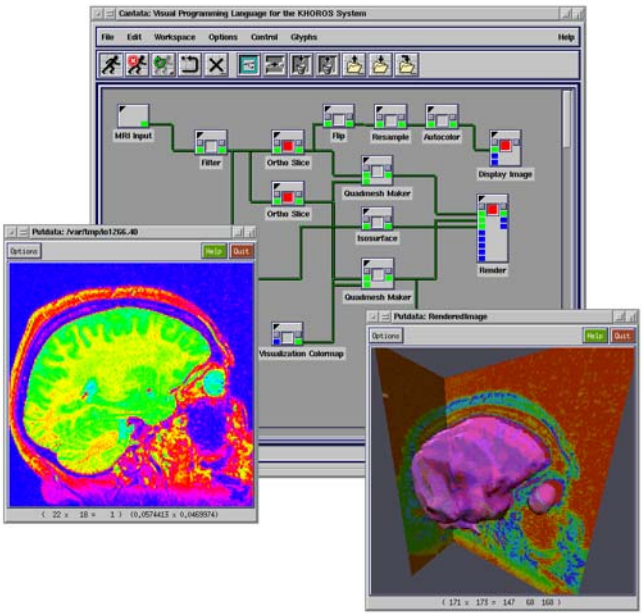

Abbildung 3-8: *Khoros* Prototyping einer NMR-Scan-Anwendung mit 2D- und 3D-Visualisierung.

Die Kosten liegen deutlich unter *AVS* und *IRIS Explorer*, allerdings ist das System mit etwa 5.000 € pro Lizenz immer noch im Hochpreisbereich angesiedelt. Auch Support, Lehrgänge o. ä. werden separat in Rechnung gestellt. Eine interessante Option für den Kunden sind die so genannten „service contracts“ – spezielle Komponenten, die nach Kundenwunsch entwickelt werden. Auf diese Art garantiert Khoral für seine Kunden immer die passende Lösung parat zu haben, ohne dass dieser selbst im Netz suchen muss oder dass das Basis-Programmpaket zu umfangreich wird. Eine kostenlose Studentenversion wird von Khoral als Sourcecode zum Download zur Verfügung gestellt, es fehlen dabei aber mehrere Entwicklungstools und Hilfsprogramme. Die Studentenversion ist nur für Unix erhältlich.

Khoros ist vor allem für den Berieb unter Unix entwickelt worden. Unter Windows ist es nur über Microsoft Interix 2.2 und den XWin32 X-Server von StarNet Communications lauffähig. Vor allem durch die X-Windows-Emulation und den dadurch verursachten Overhead ist die Performance unter Windows signifikant schlechter als unter Unix. Zudem fehlen einige unter Unix vorhandene Tools bzw. funktionieren nicht korrekt. *Khoros* beinhaltet in der Windows-Version außerdem keine Shared Libraries, da diese von Interix nicht unterstützt werden. Dies hat zur Folge, dass alle unter Unix als Shared Libraries verfügbaren Module unter Windows statisch kompiliert sind und damit die Größen der resultierenden Binaries deutlich anwachsen.

Die von Khoral angegebenen hardwaretechnischen Mindestanforderungen (Intel Pentium 100 MHz) sind als äußerst optimistisch anzusehen, selbst bei einem Pentium III mit 500 MHZ ist ein fließendes Arbeiten nicht möglich.

3.1.2.3 IRIS Explorer

Das von The Numerical Algorithms Group (NAG) Ltd. angebotene Programmpaket *IRIS Explorer* (aktuelle Version: Release 5.0; [85]-[87],[125]) ist seit etwa 12 Jahren am Markt verfügbar und zusammen mit *AVS* Marktführer. Obwohl auch *IRIS* ein kommerzielles Hochpreisprodukt ist, kann man im Netz häufig auch kostenlosen Support erhalten, so dass es mit einigem Aufwand auch einem nicht ganz unerfahrenen Einsteiger möglich ist, sich alleine in das Programm einzuarbeiten. Selbstverständlich werden aber auch hier kommerzielle Trainingskurse angeboten. Eine Studentenversion von *IRIS* ist nicht erhältlich, lediglich verbilligte Trainingskurse werden angeboten. Auf Bestellung erhält man von NAG eine dokumentierte Demoversion, die nach Installation auf einem vorher zu bestimmenden Rechner genau 30 Tage lauffähig ist.

Anders als *AVS* und *Khoros* verarbeitet *IRIS* nur das eigene Dateiformat, allerdings existieren Importmodule, die nahezu jedes Format importieren können. Support für freie Entwickler ist ebenfalls online vorhanden, entsprechende Module werden von NAG zur Verfügung gestellt. Falls das gewünschte Modul nicht vorhanden ist und man dieses auch nicht selbst implementieren will oder kann, bietet NAG auch die individuelle Entwicklung solcher Module an.

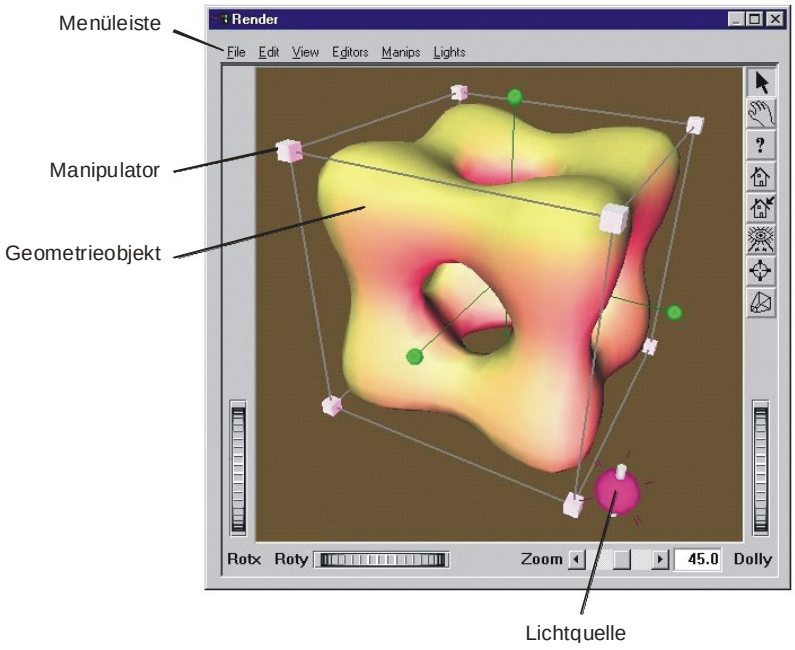

Abbildung 3-9: *IRIS Explorer* Renderwindow.

IRIS Explorer besitzt eine sehr weit entwickelte grafische Benutzeroberfläche. Das Programm ist dadurch sehr übersichtlich gestaltet und erleichtert damit auch den Einstieg. Im Gegensatz zu *AVS* und *Khoros* läuft das Programm auch problemlos unter Windows. Abbildung 3-9 zeigt das *IRIS Explorer* Renderwindow unter Windows.

Die Angabe einer Pentium-CPU mit 120 MHz als Mindestanforderung scheint allerdings auch hier bei weitem zu optimistisch, eine Taktung von 1 GHz oder mehr ist für einen flüssigen Betrieb zu empfehlen.

3.1.2.4 OpenDX

Der *Open Visualization Data Explorer*, kurz *OpenDX* ([59],[60]), wird von IBM hergestellt, die aktuelle Release ist 4.2. Das System gehört zur Gruppe der Open Source Programme und ist als Folgeprodukt aus dem eingestellten, früher kommerziell vertriebenen *IBM Data Explorer* hervorgegangen.

Es handelt sich hier um ein sehr gut dokumentiertes Werkzeug, dem ein stabiles Gerüst zugrunde liegt. Da *OpenDX* zunächst ein kommerzielles Produkt war, hat man von Anfang an für eine klare Struktur gesorgt, die leicht zu warten und zu bedienen ist. Neu erstellte Komponenten freier Entwickler werden erst nach Überprüfung der Funktionalität dem Gesamtpaket hinzugefügt, wodurch vieles an Redundanz vermieden wird und in regelmäßigen Abständen ein stimmiges Update angeboten werden kann. Dennoch ist es dem erfahrenen User freigestellt, sich im Netz mit nicht dem offiziellen Paket zugeordneten Modulen zu versorgen, so dass auch dieser beliebte Effekt der Vielfältigkeit von Open Source Produkten zur Verfügung steht. Support ist kostenfrei im Netz verfügbar und durchaus auch für Einsteiger nutzbar. Es werden außer den Spezifikationen des Systems und den grundlegenden Funktionsbeschreibungen auch viele Beispiele und Tutorials angeboten, die den Einstieg erleichtern. Ein Problem des Systems ist die ausschließliche Verwendung eines eigenen Datentyps innerhalb der Module, allerdings sind für die meisten gängigen Datenformate bereits entsprechende Importer verfügbar, die auch beim Gesamtpaket mitinstalliert werden.

Aktuell ist *OpenDX* nur auf Unix-Systemen problemlos einsetzbar. Unter Windows läuft das Programm nur unter einem X-Server, z. B. Cygwin, was jedoch durch die Unix-Emulation immer wieder zu Performance-Problemen führt. Die nötigen Installationsanleitungen sind nicht zentral erhältlich und im Gegensatz zur GUI benötigt man zur Installation unter Windows einige Erfahrung und Geduld. Das Arbeiten unter Windows ist damit Einsteigern nur möglich, wenn das Programm von erfahrenen Usern installiert wird. Die Installation unter Unix ist erwartungsgemäß weit einfacher. Ein Ansatz, der auf Plattformunabhängigkeit zielt, ist das JavaDX-Modul, welches bald im Standardpaket enthalten sein soll.

Die grafische Benutzeroberfläche ermöglicht einen leichten Einstieg und ist und intuitiv erfassbar – ein weiterer Vorteil aus der Zeit als kommerzielles Programm. Die Tutorials zeigen, dass bereits mit Hilfe einiger Mausklicks die Basisfunktionen bedienbar sind. Beschriftung und Anordnung der Bedienelemente entsprechen in der Regel dem aus anderen Programmen gewohnten Erscheinungsbild. Abbildung 3-10 zeigt den Visual Program Editor (VPE), welcher aus einer Toolbar mit verfügbaren Modulen und einer Canvas für das visuelle Erstellen und Editieren von Programmen besteht. Ein Programm ist dabei als Netzwerk einzelner Module und entsprechender Datenfluss-Verbindungen aufgebaut.

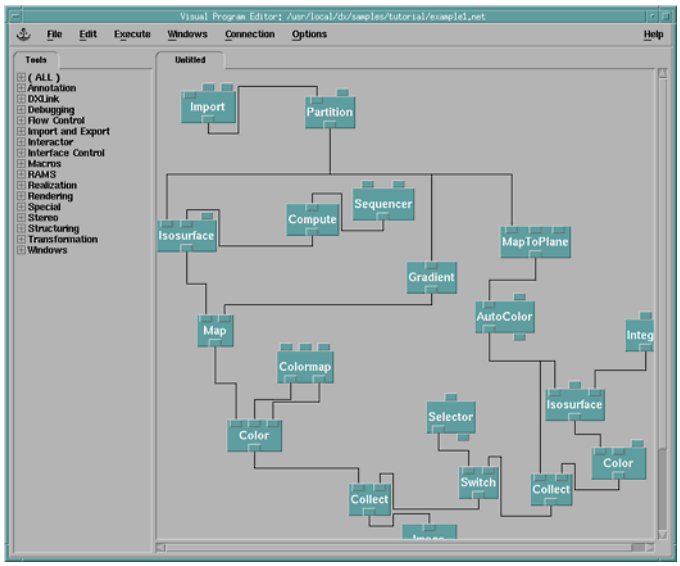

Abbildung 3-10: *OpenDX* Visual Program Editor (VPE).

Der Betrieb mit einem Pentium III mit 350 MHz ist zwar möglich, allerdings nicht so performant, dass man sinnvoll mit dem System arbeiten könnte. Für flüssiges Arbeiten sollte auf jeden Fall ein PC mit mindestens 1 GHz, 256 MB RAM sowie leistungsfähiger 3D-Grafikkarte eingesetzt werden. Für die Entwicklung interaktiver, anspruchsvoller Anwendungen ist aber auch diese Konfiguration nicht ausreichend.

3.1.2.5 inVisu PMS

Das von der epro Software GmbH hergestellte Produkt *inVisu PMS* ([29],[30]) gehört zur Kategorie der Visualisierungssysteme, die speziell für bestimmte Aufgabengebiete entworfen wurden. Es handelt sich um ein Mittelpreisprodukt, welches speziell für Aufgaben des Prozessmanagements, insbesondere Prozessüberwachung und -visualisierung, entwickelt wurde. Das 1985 vorgestellte Programm war ursprünglich für MS-DOS konzeptioniert worden und ist heute für Windows erhältlich; andere Betriebssysteme werden nicht unterstützt.

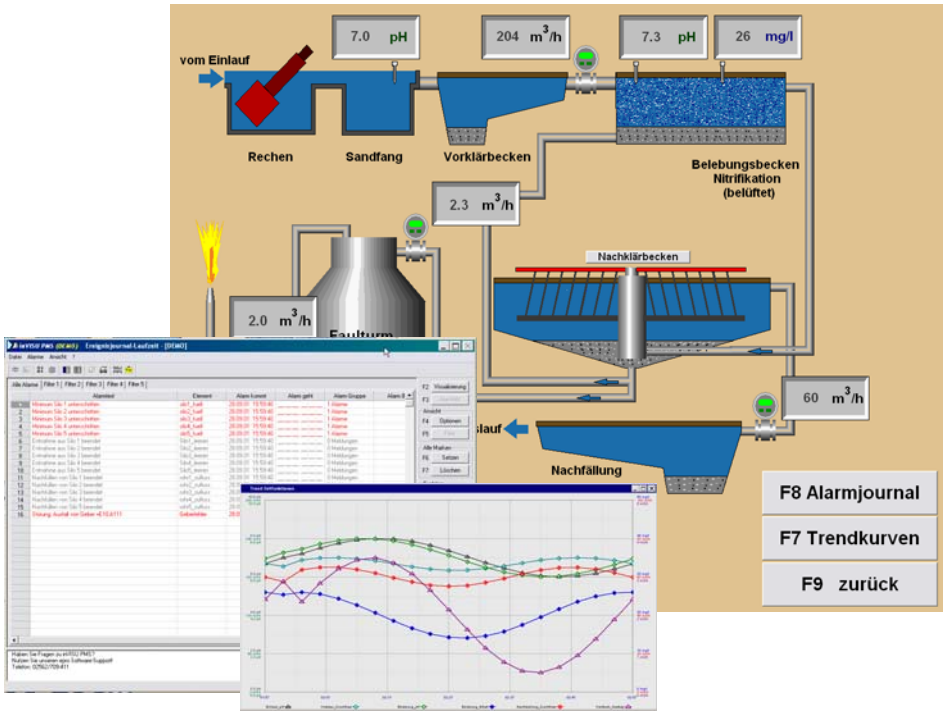

Abbildung 3-11: Visualisierung einer Kläranlage mit Ereignisjournal und Zeitfunktion.

Hauptziele des Programms sind eine hohe Benutzerfreundlichkeit sowie die Möglichkeit, Visualisierungsaufgaben ohne Programmierkenntnisse lösen zu können. Entsprechend ist die GUI von *inVisu PMS* sehr übersichtlich und verständlich gehalten, die Funktionen sind erwartungsgemäß speziell für die Industrie ausgelegt. Eine interne Weiterentwicklung des Systems durch Enduser ist nicht konkret vorgesehen, die Kommunikation mit der Außenwelt erfolgt über Standardschnittstellen wie ActiveX, OLE, DDE, OPC-Client sowie direkte Im- und Exportmöglichkeiten für Daten im CSV- oder Access-Format. Die Ausrichtung des Systems lässt sich auch gut an den vielen verfügbaren Prozesstreibern für die Kommunikation mit Steuerungen und Systembussen nachweisen. Zusätzlich zu diesen Schnittstellen existiert auch eine Anwenderprogrammschnittstelle (für C++, Visual Basic, Delphi und Java), welche den Zugriff auf die in der Online-Datenbank enthaltenen Datenelemente ermöglicht.

Mittels *inVisu PMS* können Prozessbilder und Datenmanagementsysteme, z. B. für den Anlagen- oder Maschinenbau, in der Gebäudetechnik, in der Produktion, für die Automation und Verwaltung kontinuierlicher und diskontinuierlicher Prozesse oder für Mess- und Prüfstände erstellt werden. Hierzu liefert das Programm über seine Schnittstellen kontinuierlich aktualisierte Informationen über aktuelle und historische Prozesswerte sowie Prozesszustände. Ziel ist dabei die Ermöglichung eines schnellen Überblicks über Füllstände, Durchflussmengen, Taktzeiten,

Stückzahlen und sonstige Prozessgrößen. Zudem können Benutzer über geeignete Bedien- und Steuerfunktionen in den Ablauf der angeschlossenen Anlagen oder Maschinen eingreifen.

3.1.2.6 Zusammenfassung und Bewertung

Die folgende Tabelle fasst die in den vorangegangenen Abschnitten getroffenen Bemerkungen zu den betrachteten Visualisierungssystemen in einer vergleichenden Übersicht zusammen:

	AVS	**Khoros**	**IRIS**	**OpenDX**	**inVisu**
Kommerziell	x	x	x		x
Open Source				x	
Objektorientiert	x	x	x	x	x
Verwendbare Sprachen	C, C++, Fortran	C, C++, Java, Perl	C, Fortran	C++, VB, Java, Perl	C++, VB, Java, Delphi
Basis-Klassen	x	x	x	x	x
Online-Tutorial	x	x	x	x	x
Training, Kurse	x	x	x	x	x
Unterstützte Plattformen	Unix, SGI, Sun, Windows	Unix, SGI, Sun, Windows	Unix, SGI, Sun, Windows	Unix, SGI, Sun, Windows	Windows
Minimale Systemanforderungen (lt. Hersteller)	P500; 128 MB; VGA mit 32MB	P100; 128 MB; VGA	P120; 64 MB; VGA mit OpenGL	P300; 128 MB; VGA	P300; 64 MB; VGA

Tabelle 3-1: Gegenüberstellung der untersuchten Visualisierungssysteme.

Zu obiger Tabelle muss angemerkt werden, dass die Herstellerangaben zu den minimalen Systemanforderungen extrem untertrieben scheinen. Im besten Fall läuft das Programm mit diesen Voraussetzungen, an vernünftiges oder gar flüssiges Arbeiten ist dabei nicht zu denken. Die Angaben beziehen sich natürlich immer auf das Visualisierungssystem, nicht auf die damit realisierbaren Applikationen. Für diese gelten naturgemäß gerade bei dreidimensionalen Lösungen deutlich höhere Mindestvoraussetzungen. Keines der Systeme ist dabei flexibel genug, um auf geänderte Hardwarebedingungen zu reagieren, von dynamischen Anpassungsfähigkeiten zur Laufzeit ganz zu schweigen. An den betrachteten Systemen

lässt sich auch gut erkennen, dass diese bereits lange Zeit im Einsatz sind und ursprünglich für sehr spezielle Aufgaben entwickelt wurden. Im Laufe der Zeit wurden die Systeme dann kontinuierlich erweitert – es entstanden sehr große, monolithische Systeme, deren Anforderungen an die verwendete Hardware entsprechend hoch waren. Folglich waren die Systeme nur auf performanten Unix-Workstations überhaupt lauffähig, an PC-basierte Systeme war zu diesem Zeitpunkt aufgrund der fehlenden Leistungs- und auch Grafikfähigkeit überhaupt nicht zu denken. Diese Historie spiegelt sich heute noch in den Visualisierungssystemen wider: der Betrieb unter Windows erfolgt oft nur über einen geeigneten X-Server und mit hohen Leistungs- und Stabilitätseinbußen. Um den Visualisierungssystemen einen moderneren Auftritt zu geben, wurden diesen dann nachträglich Features wie Modularität und Visual Prototyping regelrecht „aufgezwungen“. So ist oft der monolithische Kern weiter enthalten, über Schnittstellen lassen sich um diesen herum dann zusätzlich benötigte Module ergänzen. Die für eine korrekte Umsetzung eigentlich nötige Neuprogrammierung des Kerns blieb aus Zeit- und Kostengründen jedoch aus. Hierdurch ist selbst für einfachste Aufgaben wie die Visualisierung einer Kugel ein immenser Programmieraufwand erforderlich. Zudem ist jeweils die strikte Beachtung systemtechnischer Vorgaben (wie die Ableitung von systemspezifischen Basisklassen) notwendig, so dass die entwickelten Applikationen im Allgemeinen nicht einfach unter den Systemen austauschbar sind. Möchte also beispielsweise ein Wissenschaftler auf ein anderes System umsteigen oder einem Kollegen, welcher nicht auf dem gleichen System arbeitet, ein neu entwickeltes Modul zur Verfügung stellen, so ist er gezwungen, das komplette Modul neu zu implementieren und zu testen.

3.2 Einsatz von Agenten in Softwaresystemen

Techniken und Methoden der künstlichen Intelligenz haben in der Vergangenheit in vielen verschiedenen Anwendungsgebieten Einzug gefunden. Diese agentenbasierten Softwaresysteme lassen sich nach verschiedenen Kriterien klassifizieren: nach dem eingesetzten Agententyp, der verwendeten Technologien oder dem Anwendungsgebiet. Die in den Abschnitten 3.2.2 bis 3.2.5 angeführten Beispiele wurden nach letzterem Kriterium sortiert. Zunächst sollen aber einige der hierfür grundlegenden KI-Techniken vorgestellt werden.

3.2.1 Grundlegende Techniken

Die Ziele der KI-Forschung liegen weniger in der Bereitstellung irgendeiner Lösung für ein gegebenes Problem. Vielmehr steht die Frage, *wie* man dieses Problem lösen kann, im Vordergrund. Voraussetzung ist dabei, dass das Problem sowie die Bedingungen an die Lösung(en) eindeutig definiert sind. Beispielsweise

muss von Anfang an klar sein, ob das gestellte Problem zeitkritisch ist, d. h. ob nach einer gewissen Zeit eine Lösung bereitstehen muss oder ob eher eine optimale Lösung unter Vernachlässigung des Zeitaufwandes anvisiert wird.

3.2.1.1 Suchstrategien

Zentraler Punkt bei der Problemlösung mittels Suche ist die Findung einer geeigneten Abbildung der Problemstellung in eine entsprechende Datenstruktur. Ausgehend von einem initialen Zustand werden mittels einer Menge von Operatoren neue Zustände erzeugt. Dabei bezeichnet man die Kombination aus initialem Zustand und der Menge der Operatoren auch als Zustandsraum. Die Folge von Zuständen, die durch das Anwenden der Operatoren beginnend beim initialen Zustand entsteht, heißt Pfad. Das Problem ist gelöst, wenn ein solcher Pfad zu einem Zielzustand führt. Ist man nicht nur an einer beliebigen Lösung interessiert, sondern will man eine optimale, d. h. kostenminimale Lösung erreichen, so muss man zusätzlich für die Anwendung der Operatoren entsprechende Kosten über entsprechende Kostenfunktionen definieren. Ein Algorithmus ist optimal, wenn er stets die beste Lösung aus allen möglichen findet; er ist vollständig, wenn er immer eine Lösung findet, falls eine solche existiert.

Die Menge der Suchalgorithmen lässt sich in zwei Bereiche aufspalten. Verwendet der Algorithmus keine Informationen über das eigentliche Problem zur Richtungsfindung bei der Suche, so bezeichnet man diese als brute-force oder blind. Heuristische Verfahren verwenden dagegen zusätzliche Informationen, beispielsweise die Kosten oder den geschätzten Abstand zum Zielzustand.

Zur Gruppe der Brute-force-Algorithmen gehören die Breitensuche (breadth-first) und die Tiefensuche (depth-first). Beide Algorithmentypen bauen zur Suche eine hierarchische Baumstruktur auf, wobei jeder Knoten einen Zustand und jede Kante die Anwendung eines Operators repräsentiert. Der Breitensuchalgorithmus betrachtet ausgehend vom Start-Knoten alle Knoten, die direkt über eine Kante mit diesem verbunden sind. Ist keiner dieser Knoten eine Lösung, so werden alle direkt mit diesem Knoten verbundenen Knoten untersucht. Dieser Prozess wird solange fortgesetzt bis entweder eine Lösung gefunden ist oder der gesamte Baum durchsucht wurde. Der Algorithmus lässt sich wie folgt beschreiben:

```
1. Erzeuge eine Queue und füge in diese den Ausgangs-
   knoten ein.
2. Ist die Queue leer, so beende den Algorithmus. In
   diesem Fall gibt es keine Lösung.
3. Entferne den ersten Knoten k der Queue.
4. Ist k eine Lösung, so beende den Algorithmus mit
   Lösung k.
5. Füge alle Söhne von k am Ende der Queue ein.
6. Gehe zu 2.
```

Die Breitensuche ist vollständig, aber im Allgemeinen nicht optimal. Der Aufwand in Bezug auf Zeit und Speicherplatz wächst exponentiell.

Der Tiefensuchalgorithmus ist identisch dem Breitensuchalgorithmus, als einzigen Unterschied fügt er im fünften Schritt die Söhne nicht am Ende, sondern am Anfang der Queue ein (genau genommen wird hier also keine Queue, sondern ein Stack verwendet). Hierdurch durchläuft er einen Pfad immer von der Wurzel bis zu einem Blatt. Auch die Tiefensuche ist nicht optimal. Da hier zudem Schleifen vorkommen können, ist der Algorithmus nicht vollständig. Im Vergleich zur Breitensuche ist hier jedoch der Speicheraufwand geringer.

Im Gegensatz zu den gerade beschriebenen Algorithmen verwenden die heuristischen Suchmethoden zusätzliche Informationen über das gegebene Problem, um die Wahrscheinlichkeit, dass ein Pfad zu einer Lösung führt, zu bestimmen. Als Beispiel soll hier die Best-First-Suche kurz beschrieben werden.

Die Best-First-Strategie unterscheidet sich von den Brute-Force-Algorithmen vor allem dadurch, dass für neu in die Queue einzufügende Knoten zunächst eine Kostenfunktion berechnet wird. Anhand des Ergebnisses wird der Knoten dann in die Queue einsortiert. Bei der so genannten Greedy-Suche (greedy = gierig) wird hier immer der Knoten mit den kleinsten Kosten an den Anfang der Queue gestellt. Da die exakten Kosten im Vorhinein in der Regel nicht berechnet werden können, müssen sie mittels einer entsprechenden heuristischen Funktion abgeschätzt werden. Der Greedy-Algorithmus verhält sich ähnlich der Tiefensuche, seine Qualität hängt entscheidend von der Wahl der Kostenfunktion ab.

3.2.1.2 Wissensrepräsentation

Ein wichtiges Forschungsgebiet der KI stellt die Repräsentation von Wissen dar. Dabei stellt sich natürlich zuerst die Frage, was Wissen überhaupt ist. Der Mensch erlangt sein Wissen durch Erfahrung, indem er über seine natürlichen Sinne seine Umwelt wahrnimmt. Im Rechner dagegen verwendet man Symbole, um Wissen intern zu repräsentieren und zu manipulieren. Unter einem Symbol versteht man dabei eine Zahl oder einen String (bestehend aus einzelnen Zeichen), der ein Objekt entsprechend repräsentiert. Da diese Symbole in der Regel für den Menschen relativ schwer zu verstehen sind, benötigt man zur externen Wissensrepräsentation zusätzlich eine Abbildung von den Symbolen in eine Form, die besser auf den Menschen zugeschnitten ist und umgekehrt. Im Folgenden werden einige Ansätze zur Wissensrepräsentation im Rechner kurz vorgestellt.

Eine sehr gebräuchliche Form der Wissensrepräsentation ist das so genannte *prozedurale Wissen*. Dabei sind in prozeduralem Code nicht nur die Fakten, sondern auch der Ablauf der Operationen für die Benutzung und die Veränderung dieser Fakten enthalten. Hierzu werden keine besonderen Datenstrukturen oder Objekte

verwendet, der gesamte Programmcode selbst stellt die Wissensrepräsentation dar. Es ist offensichtlich, dass die Umsetzung prozeduralen Wissens sehr einfach mit gängigen Skriptsprachen möglich ist. Entscheidender Nachteil ist jedoch die enge Verknüpfung von Wissen und dessen Behandlung. Um dies zu vermeiden liegt es nahe, das eigentliche Wissen (Fakten, Regeln usw.) und den es verwendenden prozeduralen Code getrennt zu speichern, wodurch auch das nachträgliche Verändern des Wissens wesentlich erleichtert wird. Diese Form der Wissenrepräsentation bezeichnet man als *deklarativ.*

Eine weitere Möglichkeit zur Wissenspeicherung stellen *relationale Datenbanken* dar. Wissen wird hier in Form von Tupeln repräsentiert, wobei jedes Tupel eine Menge von Spalten oder Feldern mit den entsprechenden Werten bzw. Eigenschaften des betrachteten Objektes enthält. Die Tupel können dann über entsprechende Sprachen angesprochen oder verändert werden, als Beispiel sei hier die Structured Query Language (SQL) genannt. Die relationale Wissensrepräsentation ist sehr flexibel, jedoch lassen sich hiermit die in der realen Welt häufig vorkommenden komplexen Beziehungen zwischen verschiedenen Objekten nur schwer abbilden.

In den bisher vorgestellten Methoden wird Wissen einzeln abgespeichert ohne dabei Zusammenhänge und Gemeinsamkeiten zu betrachten. Hierarchische Repräsentationen dagegen gruppieren ihr Wissen in Objektklassen, die nach Typ und Abstraktionsstufe der jeweiligen Objekte angeordnet werden. Beispielsweise wäre im Bereich der Rechnerhardware der allgemeine Typ „Prozessor“ über den Typen „Intel 486“, „Intel Pentium“ und „AMD Duron“ angeordnet, während die Typenverwandtschaft durch entsprechende Verbindungen dargestellt wäre. Schlussfolgerungen werden durch eine solche hierarchische Ordnung sehr erleichtert, da zunächst nur die allgemeinen Klassen betrachtet werden müssen und anschließend nur in den passenden Kategorien nach der Lösung gesucht werden muss. Diese Form der Wissensrepräsentation lässt sich hervorragend mit Hilfe objekt-orientierter Programmiersprachen wie C++ oder Java im Rechner abbilden.

Oftmals möchte man repräsentiertes Wissen auch zwischen unterschiedlichen Systemen austauschen, die jedoch in der Regel jeweils eigene, nicht miteinander kompatible Wissenstrukturen verwenden. Hierzu benötigt man eine gemeinsame Sprache zum Austausch der Informationen. Einen Standard definiert in diesem Zusammenhang das in Abschnitt 2.2.4.4 beschriebene Knowledge Interchange Format (KIF).

3.2.1.3 Reasoning Systeme

Die im vorhergehenden Abschnitt vorgestellten Wissenrepräsentationen werden vor allem in so genannten Reasoning- oder Case-based Reasoning- (CBR, Fallbasiertes Schließen) Systemen eingesetzt.

Die wohl bekannteste Form der Wissensrepräsentation und -auswertung sind hier die If-Then-Regeln:

```
if <Bedingung> then <Anweisung>
```

Jede Regel stellt für sich eine eigenständige Beschreibung eines gewissen Fakts der Wissensdatenbank dar und führt bei Erfülltsein ihrer Bedingung die zugehörige Anweisung aus. Diese Anweisung kann dabei entweder wieder neues Wissen erzeugen oder einer Behauptung einen Wahrheitswert zuordnen. Aufgrund ihres einfachen und logischen Aufbaus sind If-Then-Regeln leicht verständlich. Das Hinzufügen von Wissen durch neue Regeln in der Wissensdatenbank als auch die Modifikationen bestehenden Wissens ist problemlos zu bewerkstelligen. Mit wachsender Zahl der Regeln in einer solchen Datenbank geht jedoch der Vorteil der leichten Verständlichkeit durch fehlende Übersichtlichkeit sukzessive verloren. Hier bietet sich die Aufteilung der Regeln in logische Gruppen an.

Die Auswertung der Regeln erfolgt dann durch ein entsprechendes Reasoning System, welches häufig auch als Expertensystem oder Wissensbasiertes System bezeichnet wird. Ein solches System besteht aus drei elementaren Komponenten: einer Wissensdatenbank, einem Arbeitsspeicher bestehend aus bekannten oder abgeleiteten Fakten und Daten und einer Inferenzmaschine, die die Logik zur Bearbeitung der Regeln und Daten enthält. Im Folgenden sollen zwei unterschiedliche Ansätze zur Auswertung von Regeln kurz vorgestellt werden: *forward chaining* zur Erzeugung neuer Fakten aus bekannten Fakten sowie *backward chaining* zur Auswertung einer Aussage nach wahr oder falsch.

Der Forward chaining-Algorithmus erzeugt neue Fakten durch direkte Anwendung der in der Wissensdatenbank vorhandenen Regeln auf bekannte Fakten und ist hierdurch sehr schnell. Er findet daher auch häufig in zeitkritischen Systemen wie Echtzeit-Überwachungs- und Diagnosesystemen Verwendung. Beispielsweise kann ein Benutzer einem entsprechenden System technische Daten und Aussagen über den in seinem Rechner verwendeten Prozessor als initiale Informationen eingeben und erhält dann als Ergebnis den Prozessortyp zurückgeliefert. Der Algorithmus lässt sich mit den folgenden Schritten beschreiben:

1. Lade die vorhandenen Regeln in die Inferenzmaschine und die bekannten Fakten und initialen Daten in den Arbeitspeicher.
2. Vergleiche die Regeln (bzw. deren Bedingungen) mit den Daten im Arbeitsspeicher und bestimme hieraus die Menge der anwendbaren Regeln. Diese Menge bezeichnet man auch als Konfliktmenge (*conflict set*).
3. Bestimme mit einem Konfliktlösungsverfahren genau eine Regel aus dem conflict set.

4. Führe die in 3. bestimmte Regel aus, wodurch neue Fakten generiert werden können.
5. Gehe zu 2 bis die Konfliktmenge leer ist.

Ein zentraler Punkt des Algorithmus stellt dabei das verwendete Verfahren zur Konfliktlösung dar. Ist das conflict set leer oder besteht nur aus einer Regel, so ist das Problem trivial. Enthält es jedoch mehrere anwendbare Regeln, so existieren mehrere Alternativen zur Auswahl einer anzuwendenden Regel. Die einfachste Möglichkeit ist sicherlich die Verwendung der ersten Regel aus der Menge. Trotzdem liefert dieser Ansatz für viele Anwendungen ausreichende Ergebnisse. Die folgenden Kriterien berücksichtigen dagegen Informationen über bekannte Fakten bzw. über die vorangegangenen Ausführungsschritte:

- Verwende die charakteristischste Regel, d. h. wähle diejenige Regel mit den meisten Bedingungen aus (da die Regel bereits im conflict set enthalten ist, sind ja alle Bedingungen erfüllt).
- Wähle die Regel, die sich auf die zuletzt geänderten Daten bezieht. Um dieses Verfahren anwenden zu können, müssen die Daten im Arbeitsspeicher entweder nach der Zeit der letzten Änderung geordnet werden bzw. einen entsprechenden Zeitstempel mit sich führen.
- Führe keine Regel aus, die im vorhergehenden Durchlauf bereits verwendet wurde.
- Enthält nach Anwendung obiger Auswahlkriterien das conflict set noch mehr als eine Regel, so wähle per Zufall eine dieser Regeln aus.

Zusätzlich zu den genannten Alternativen werden den Regeln häufig schon bei deren Definition Prioritäten zugeordnet, wodurch die Anzahl der zu überprüfenden bzw. anwendbaren Regeln stark verkleinert werden kann.

Während beim forward chaining aus einer initialen Menge von Fakten durch Anwendung von Regeln ein vorher nicht bekannter Zielzustand erreicht wird, stellt beim backward chaining dieser Zielzustand den Ausgangspunkt der Betrachtung dar. Ziel ist hier, durch die Verwendung der Regeln zu bestimmen, ob die zugehörige Aussage wahr oder falsch ist. Anwendung findet das backward chaining zum Beispiel in „beratenden“ Experten-Systemen, d. h. Systemen, bei denen der Benutzer eine Frage stellt, die das System durch Stellen weiterer Fragen versucht zu beantworten. Im Beispiel würde dies bedeuten, dass der Benutzer dem System die Frage „Ist mein Prozessor ein Intel Pentium?“ stellen könnte und dieses dann über entsprechende Detailfragen zum Ergebnis „ja“ oder „nein“ kommt. Der Backward chaining-Algorithmus gestaltet sich wie folgt:

1. Lade die vorhandenen Regeln in die Inferenzmaschine und die bekannten Fakten und initialen Daten in den Arbeitspeicher.

2. Spezifiziere die Zielvariable.
3. Bestimme die Menge der Regeln, die bei Anwendung die Zielvariable verändern können, und lege diese Regeln jeweils auf den Ziel-Stack.
4. Ist der Ziel-Stack leer, halte an.
5. Betrachte die oberste Regel des Ziel-Stacks.
6. Teste, ob die betrachtete Regel anwendbar ist. Teste hierzu die einzelnen Bedingungen dieser Regel wie folgt:
   - Ist die betrachtete Bedingung wahr, so überprüfe die nächste Bedingung.
   - Ist die Bedingung falsch, so nimm die betrachtete Regel vom Stack. Mache weiter mit 4.
   - Ist der Wahrheitswert der Bedingung nicht bestimmbar, weil die zugehörige Variable nicht bekannt ist, so gehe zu 3 mit der unbekannten Variablen als neuer Zielvariablen.

   Sind alle Bedingungen der Regel erfüllt, so führe sie aus. Nehme die Regel vom Stack. Mache weiter mit 4.

Bisher wurden in den Überlegungen nur die boolesche oder binäre Logik betrachtet. Oftmals lassen sich Aussagen in der Praxis jedoch hiermit nicht ausreichend beschreiben, da ihrem Zutreffen eine gewisse Wahrscheinlichkeit zugrunde liegt. Diesen Sachverhalt berücksichtigt die so genannte Fuzzy-Logik. Im Gegensatz zur booleschen Logik unterscheidet man hier nicht nur die beiden Zustände wahr und falsch, sondern arbeitet mit Wahrheitswerten, die zwischen 0.0 (d. h. falsch) und 1.0 (d. h. wahr) liegen können. Hiermit können dann auch Aussagen wie sehr wahrscheinlich wahr (0.9) oder eventuell nicht zutreffend (0.2) beschrieben werden. Durch die Verwendung der Fuzzy-Logik können auch die Regeln der Wissensdatenbank natürlicher formuliert werden, da man nicht alle Aussagen auf Wahrheitswerte abstrahieren muss. In einem Reasoning System, das auf Fuzzy-Logik aufbaut, erfolgt das Schließen mittels forward chaining. Im Gegensatz zum oben beschriebenen Algorithmus erfolgt jedoch in den Schritten 2 und 3 keine Bestimmung der anwendbaren Regeln mit anschließender Auswertung der Konfliktmenge. Stattdessen müssen alle Regeln ausgewertet werden, da jede von ihnen zu einem bestimmten Grad wahr sein kann. Die jeweiligen Wahrheitswerte müssen dann zur Erzeugung eines entsprechenden Ausgabewertes geeignet kombiniert werden.

3.2.2 Industrielle Anwendungen

Schon relativ früh fand die Agententechnologie Einzug in viele industrielle Anwendungen. Ein Beispiel hierfür ist die *Prozesssteuerung und -überwachung*. Der Einsatz intelligenter Agenten ist hier nahe liegend, da die involvierten Prozess-Controller selbst ebenfalls autonome reaktive Systeme sind. Ein bekanntes

Beispiel einer Softwareplattform für die Generierung von verteilten Multi-Agenten-Systemen und entsprechender Applikationen ist das von Jennings entwickelte System ARCHON (ARchitecture for Cooperative Heterogeneous ON-line systems; [16],[62],[63]). Dieses wurde u. a. für das Management von Elektrizitätsverteilung, zur Kontrolle von Partikelbeschleunigern und in der Robotik eingesetzt.

Das agentenbasierte System YAMS (Yet Another Manufactoring System) wurde für die *industrielle Herstellung und Produktion* entwickelt und wendet das so genannte Contract Net Protocol [107] auf die Produktionsüberwachung an. Der Produktionsprozess stellt in der Regel eine sehr komplexe Aufgabe dar. Ein Unternehmen besteht im Allgemeinen aus mehreren Produktionsbetrieben, die jeweils in verschiedene Abteilungen aufgeteilt sind, welche zusätzlich flexibel bezüglich ihrer Funktionalität im jeweiligen Produktionsprozess gruppiert sind. YAMS modelliert alle Firmen und deren Komponenten als einzelne Agenten mit entsprechend zugeordneten Leistungsvermögen. Mittels des Contract Net Protocols werden die einzelnen Aufgaben dann top-down durch die Unternehmenshierarchie bis zum verantwortlichen Abteilungsagenten geleitet.

Ein weiteres Beispiel für den Einsatz von Agenten findet sich im Bereich der *Flugüberwachung*, in welchem sich die Modellierung der einzelnen autonomen Komponenten der „echten Welt“, d. h. sowohl Flugzeuge als auch Flugüberwachungssysteme, durch intelligente Agenten anbietet. Ziel des agentenbasierten Systems OASIS (Optimal Aircraft Sequencing using Intelligent Scheduling; [75]) ist die maximale Ausnutzung von Rollfeldern. Hierzu werden die landenden Flugzeuge zunächst in eine optimale Reihenfolge gebracht, entsprechende Landezeiten zugeordnet und nachfolgend in Echtzeit überwacht. Das gesamte Flugmanagement wurde dabei in Teilaufgaben zerlegt und jeder Aufgabe ein eigener, unabhängiger Agent zugeordnet. Die einzelnen Agenten kommunizieren zur Lösung der Gesamtaufgabe untereinander asynchron mittels Nachrichten. Die Agenten lassen sich in zwei Klassen unterteilen: zum einen in die so genannten globalen Agenten zur Flugzeug-übergreifenden Verwaltung und Entscheidungsfindung, zum anderen in die Flugzeug-individuellen Flugzeug-Agenten. Die Reasoning-Komponenten basieren dabei auf dem echtzeitfähigen Protocol Reasoning System (PRS).

3.2.3 Kommerzielle Anwendungen

Bereits in Abschnitt 2.1 wurde im Kontext der Scientific Visualization auf das rapide Anwachsen der Datenmengen bei numerischen Simulationen und die hierdurch aufkommenden Problematiken eingegangen. Doch auch im täglichen Leben zeichnet sich eine analoge Entwicklung ab: durch den Einzug der neuen Medien in immer mehr Bereichen nimmt die Flut an Informationen immer mehr zu. Zusätzlich ist es oftmals schier unmöglich, die Suche nach einer gewissen Information geeignet einzugrenzen, um der Vielfalt zumindest in gewissem Rahmen Herr zu

werden. Entsprechend werden geeignete Hilfsmittel zum *Informations-Management* benötigt. Diese müssen vor allem zwei Hauptforderungen gerecht werden. Zum einen müssen alle Informationen, die den gesetzten Anforderungen entsprechen, in der Regel an vielen verschiedenen Stellen gesammelt werden. Zum anderen muss der Fokus auf die wirklich benötigten Daten durch geeignete Filterung gewahrt bleiben. Der Erfolg beim Einsatz entsprechender Tools hängt jedoch weiterhin stark vom jeweiligen Benutzer ab, da dieser den Managementprozess selbst aktiv lenken muss. Hier bietet sich der Einsatz von geeigneten, autonomen Agenten zur Suche und Filterung von Informationen im WWW im Auftrag des Benutzers geradezu an.

Ein Beispiel hierfür stellt der Email-Filter-Agent Maxims [77] dar. Der Agent „beobachtet“ den Benutzer bei der Arbeit mit dem Email-Reader und lernt somit die entsprechenden Arbeitsweisen beim Erhalt bestimmter Emails. Basierend auf seinem Wissen kann der Agent dann Vorhersagen darüber treffen, was ein Benutzer mit einer Email machen wird (löschen, beantworten, archivieren usw.) und ihm geeignete, unterstützende Vorschläge unterbreiten.

Ein weiteres Beispiel für den Einsatz von Agenten stellt die Automatisierung von Teilen des *elektronischen Handelsverkehrs* dar [78]. Entsprechend den Vorgaben des Benutzers können so „kaufende Agenten“ im WWW nach geeigneten virtuellen Verkaufräumen suchen und mit den „verkaufenden Agenten“ in Verhandlung treten. Je nach Aufgabenstellung und Autonomitätsgrad kann der „kaufende Agent“ somit für den Benutzer Informationen über entsprechende Produkte sammeln und auswerten oder gar den Auftrag in dessen Namen tätigen. Der elektronische Einkauf untergliedert sich dabei in sechs grundlegende Stufen: Identifizierung eines Bedürfnisses, Auswahl geeigneter Produkte, Auswahl möglicher Händler, Verhandlungen, Kauf / Auslieferung, sowie Evaluierung / Kundendienst.

Ein Beispiel für einen solchen „elektronischen Marktplatz“ gibt das Online-Transaktionssystem Kasbah [15], das aus kaufenden und verkaufenden Agenten besteht, die Verkäufe durch entsprechende Interaktionen untereinander abwickeln. Der Benutzer erzeugt hier einen kaufenden oder verkaufenden Agenten, gibt diesem dessen strategische Richtung (Startpreis, niedrigster bzw. höchster akzeptierbarer Preis, Zeitraum, in dem der Kauf getätigt sein muss, Verhandlungsstrategie usw.) vor und schickt ihn dann auf einen elektronischen Marktplatz. Verhandlungen sind in Kasbah relativ einfach realisiert: der kaufende Agent schlägt einen Preis vor, der verkaufende Agent antwortet auf das Gebot mit „Ja“ oder „Nein“. Bei Ablehnung des Angebots erhöht der kaufende Agent gegebenenfalls sein Angebot. Die Erhöhung erfolgt dabei entsprechend der vorher vom Benutzer ausgewählten Verhandlungsstrategie, modelliert durch eine lineare, quadratische bzw. exponentielle Zeit-Preis-Funktion. Es zeigte sich bei entsprechenden Untersuchungen, dass die einfache und übersichtliche Verhandlungsheuristik u. a. zu einer gesteigerten Kun-

denakzeptanz für den Einsatz des Systems bei elektronischen An- oder Verkäufen führt.

3.2.4 Medizinische Anwendungen

Intelligente Agenten und KI-Techniken werden zunehmend auch in medizinischen Anwendungen eingesetzt. Im Folgenden sollen hier Anwendungen in den Bereichen Patientenüberwachung und medizinische Betreuung vorgestellt werden.

Guardian ([52],[73]) ist ein experimentelles System zur Unterstützung bei der Überwachung von an medizinischen Geräten angeschlossenen Patienten auf der Intensivstation. Aufgrund der Menge der anfallenden Daten und möglichen Behandlungsmethoden sowie Einstellmöglichkeiten der entsprechenden medizinischen Geräte sind hier selbst erfahrene Ärzte oftmals überfordert. Zentrale Zielsetzung von Guardian ist daher eine intelligente, effektive und vor allem auch zuverlässige Überwachung der Behandlung von verschiedensten medizinischen Situationen. Das Gesamtsystem unterteilt sich hierzu in drei agentengestützte Bereiche: Informationsgewinnung, Reasoning und Ausführung von zugehörigen Maßnahmen. Als Basis für die Reasoning- und Maßnahmeprozesse dient dabei die Erfassung und Sammlung der von der medizinischen Umgebung gelieferten Messwerte und Informationen. Diese Daten werden anschließend im wissensbasierten Reasoning-Prozess ausgewertet und mittels Prognose- und Planungstechniken geeignete, die medizinische Umgebung beinflussende Aktionen ausgeführt. Die zur Lösung des Gesamtproblems nötige Kommunikation unter den einzelnen Agenten und Bereichen erfolgt dabei durch den Austausch von Informationen und Wissen über ein Blackboard.

Bei der medizinischen Betreuung eines Patienten sind in der Regel verschiedene Personen und Institutionen involviert, deren Arbeit und Entscheidungen entsprechend koordiniert werden müssen. Dieser Managementprozess sowohl zwischen als auch innerhalb den verschiedenen Einrichtungen wird heute in der Regel durch geeignete Softwaresysteme unterstützt. Die Informationen sind dabei physikalisch verteilt und der Zustand der Umgebung ist dynamisch und nicht vorhersagbar. Zusätzlich müssen Entscheidungen häufig auf Basis unvollständiger Informationen getroffen werden. Ein agentenbasierter Ansatz wie in [56] bietet sich daher idealerweise an. Voraussetzung hierfür ist die Erfüllung einiger zentraler Bedingungen. So müssen alle eingesetzten Agenten eine gemeinsame Sprache sprechen und Mechanismen und Strukturen für die Übertragung von Aufgaben an die jeweils am besten geeigneten Agenten definiert werden. Weiterhin müssen die einzelnen Agenten auch aus unvollständigen oder gar widersprüchlichen Informationen passende Schlussfolgerungen ziehen und trotz der dynamischen und unvorhersehbaren Umgebungen ihre Ziele und Pläne effizient weiterverfolgen. Der in Prolog implementierte Prototyp ist aus mehreren Schichten aufgebaut, unter ande-

rem aus einem wissensbasierten System, einem Mensch-Maschine-Interface sowie einem Kommunikationsmanager. Um sowohl komplexe Aufgaben erfüllen und gleichzeitig auch innerhalb eines angemessenen Zeitraums auf neue Informationen reagieren zu können, kommen dabei sowohl reaktive als auch proaktive Agenten zum Einsatz, die untereinander nachrichtenbasiert kommunizieren.

3.2.5 Anwendungen in der Unterhaltungsindustrie

Auch in der Unterhaltungsindustrie lassen sich Agentensysteme nicht mehr wegdenken und werden in immer mehr Spielen integriert. Die grundlegende Idee ist hier, dass die Story oder der Ablauf eines Computerspiels nicht jedes Mal gleich ist, sondern dynamisch durch die Handlungen des Spielers bestimmt und verändert wird. Damit die Computerfiguren überzeugend auf den Betrachter wirken, müssen diese neben der Interaktion mit dem Spieler zusätzlich auch untereinander interagieren. Umgesetzt werden solche Agenten häufig mittels regelbasierten Programmiersprachen wie der in [127] vorgestellten Sprache RTA. Die aus Regeln bestehenden RTA-Programme werden dabei in asynchrone Digitale Logik Schaltkreise (bestehend aus Registern, logischen Elementen und Delays) übersetzt, wodurch sich die entsprechenden Agenten durch eine gerade in interaktiven Spielen sehr wichtige hohe Reaktivität und Echtzeitfähigkeit auszeichnen.

3.3 Kontextsensitive Visualisierung

Kontextsensitiv (oder kontextbezogen / kontextadaptiv) bedeutet sinngemäß „auf den (aktuellen) Zusammenhang – den Kontext – bezogen“. In der Informatik versteht man unter kontextsensitiv allgemein die Eigenschaft eines (Programm-)Systems, sich auf die aktuell vorliegende Situation zu adaptieren.

Das Ziel dieser Arbeit liegt in der Übertragung des Begriffes kontextsensitiv auf den Bereich der Visualisierung zur Sicherstellung effizienter und kontextbezogener Visualisierungsapplikationen. Die dieser Arbeit zugrunde liegende Definition der kontextsensitiven Visualisierung lautet daher wie folgt:

Eine Visualisierung bzw. eine Visualisierungsapplikation ist genau dann kontextsensitiv, wenn sie vorgegebene Kontexte berücksichtigt und flexibel auf sich dynamisch ändernde Situationen proaktiv (ohne Notwendigkeit eines Benutzereingriffs) reagiert.

Im Folgenden werden zunächst die zentralen Szenarien der kontextsensitiven Visualisierung identifiziert. Schon hier sei bemerkt, dass sich die einzelnen identifizierten Kontexte nicht gegenseitig ausschließen, sondern dass sich im Gegenteil die Bereiche untereinander beeinflussen. Zudem ist jeweils ein starker Bezug zur

Visualisierungspipeline vorhanden, welcher in diesem Rahmen als visualisierungstechnischer Kontext aufgefasst wird. Anhand der Definition des Begriffes „kontextsensitive Visualisierung“ lässt sich bereits erkennen, dass die Umsetzung einer kontextsensitiven Visualisierung durch die innovative Kopplung von Visualisierungspipeline und Agententechnologie gelingen kann. Basierend auf den identifizierten Szenarien wird daher eine agentenbasierte Visualisierungskontrolle durch intelligente Überwachung und Regelung der Visualisierungspipeline vorgestellt.

3.3.1 Szenarien

3.3.1.1 Benutzerkontext

In der heutigen Informationsgesellschaft stellen Individualität und Personalisierung entscheidende Faktoren für den Erfolg oder Misserfolg einer Idee oder eines Produktes dar. Beide Begriffe lassen sich unter dem Begriff „Benutzerkontext“ zusammenfassen.

Ein Benutzer wird durch ein Benutzermodell beschrieben, welches die individuellen Benutzereigenschaften repräsentiert, z. B. Annahmen, Interessen, Präferenzen, Fähigkeiten, Aufgaben usw. Diese Eigenschaften sind entweder explizit vom Benutzer vorgegeben (Benutzerprofil) oder implizit aus dem Benutzerverhalten abgeleitet. Der Kontext eines durch ein Benutzermodell beschriebenen Benutzers umfasst physikalische Informationen (Position im physikalischen oder virtuellen Raum und Bewegungen darin), soziale Kontexte (Stellung, Rollen, Berechtigungen), Zeit (Termine, Zeitvorgaben), individuelle Fähigkeiten und Interessen.

In heute existierenden Visualisierungsanwendungen ist jedoch die Umsetzung des Benutzerkontext-Paradigmas nur selten und dann auch nur rudimentär zu finden. In der Regel sind individuelle Anpassungen auf einige wenige Merkmale wie die Optik der Benutzeroberfläche oder den Zuschnitt bestimmter Programmversionen auf gewisse Benutzergruppen beschränkt. Die hierdurch entstehenden Einschränkungen machen so beispielsweise oft die Einarbeitung in Anwendungen nötig, welche für die Fähigkeiten und Ziele des jeweiligen Benutzers bei weitem zu komplex und umfangreich konzeptioniert sind.

3.3.1.2 System- und Interaktionskontext

Auf dem Markt befindet sich heutzutage eine Vielzahl von sich hardwaretechnisch stark unterscheidenden Plattformen. Allein im PC-Bereich finden sich in den Haushalten bedingt durch die rapide fortschreitenden Entwicklungen die unterschiedlichsten Rechnerkonfigurationen und -leistungsstufen. Die einzelnen Fähigkeiten eines Rechners (wie Prozessorleistung, Speicherausbau, Grafikfähigkeiten,

Betriebssystem usw.) lassen sich unter dem Begriff „Systemkontext“ zusammenfassen.

Neben den statischen Parametern des Systemkontextes müssen in diesem Zusammenhang aber auch sich dynamisch ändernde Werte in Betracht gezogen werden. So ändert sich je nach Art und Anzahl der parallel oder (unbemerkt) im Hintergrund laufenden Programme die Auslastung des Systems und damit die für die jeweilige Applikation zur Verfügung stehenden Systemressourcen. Auf Multi-User-Rechnern sind diese zusätzlich von der Anzahl der aktiven Benutzer abhängig. Entsprechend kann sich die Reaktionszeit bei Interaktionen des Benutzers in beiden Fällen zum Teil signifikant verändern. Die dynamischen Systemparameter werden daher unter dem Begriff „Interaktionskontext“ zusammengefasst.

Wie bereits in Abschnitt 3.1 beschrieben, wurden die meisten sich zurzeit auf dem Markt befindlichen Visualisierungssysteme bei Ihrer Entstehung für spezielle Anwendungszwecke konzipiert und erst im Nachhinein erweitert. Daher zeigen sie vor allem Mängel bei der Flexibilität des Visualisierungsprozesses. Dies bedeutet, dass sie je nach Applikation oder Konfiguration entweder eine hohe Darstellungsqualität mit eingeschränkten Interaktionsmöglichkeiten oder eine Echtzeitvisualisierung einhergehend mit einer verringerten Renderingqualität anbieten. Zudem sind die Systeme oft nicht oder nur eingeschränkt auf verschiedenen Plattformen und Hardwarekonfigurationen lauffähig. Außerdem muss der Benutzer die Abgleichung zwischen Framerate und Darstellungsqualität manuell einstellen, indem er die notwendigen Steuerparameter von Hand ändert oder die verursachenden Systemkomponenten identifiziert und deren Programmcode für die aktuell vorliegende Situation modifiziert bzw. optimiert. Beide Aufgaben kann er jedoch nur bewältigen, wenn er tief greifende Kenntnisse der einzelnen Parameter und gegebenenfalls ausreichende Programmiererfahrung besitzt. Hinzu kommt, dass schon bei einfachen Applikationen die Zahl der beeinflussbaren Parameter in der Regel sehr groß ist und diese sich zudem häufig gegenseitig beeinflussen. Die geeignete manuelle Justierung ist daher mit einem hohen Zeitaufwand verbunden, das Finden einer zufrieden stellenden Konfiguration ist oftmals Glücksache. Vor allem ist aber auf diesem Wege eine dynamische Anpassung zur Laufzeit, z. B. an sich ändernde Systemlasten, so gut wie unmöglich.

3.3.1.3 Situationskontext

Der Situationskontext ist eng mit dem Benutzerkontext verbunden und beschreibt die aktuelle Situation des Benutzers. Zentrale Parameter stellen dabei Orts- und Umgebungsmerkmale dar, welche insbesondere die Frage, wo sich der Benutzer gerade befindet, beantworten. Gerade im Hinblick auf die heute immer wichtiger werdende Mobilität stellt die Berücksichtigung der aktuellen Situation eines Benutzers einen äußerst zentralen Faktor für die Darstellung von Informationen

dar. Location Based Services (LBS) stellen standortbezogene Dienste zur Verfügung und bieten dem Benutzer spezifische, auf seinen jeweiligen Raumkontext zugeschnittene Dienste an. So erwartet ein Benutzer eines Städteinformationssystems zum Beispiel für die Vorbereitung seiner Reise eine Auflistung aller Sehenswürdigkeiten mit einer kurzen Beschreibung, vor Ort aber Informationen zu einzelnen Details des gerade besuchten Objektes.

Weitere Parameter ergeben sich beispielsweise aus den Fragen „Dienstlich oder privat?“, „Allein oder mit mehreren?“, „Viel Zeit oder wenig Zeit?“ usw. So ändert sich die Sichtweise auf ein und dasselbe Objekt teilweise drastisch, wenn man es aus beruflicher oder privater Sichtweise betrachtet. Auch sollte die Darstellung vertraulicher Informationen, wie zum Beispiel Kontostände, situationsabhängig an- bzw. abschaltbar sein.

3.3.1.4 Darstellungskontext

Neben den für die Performance interessanten Hardware- und Auslastungsaspekten ist vor allem die Art des Ausgabemediums für Visualisierungsanwendungen von großer Bedeutung. Je nach Art, Größe und Auflösung muss die darzustellende Information verschieden aufbereitet werden, als Präsentationstypen können z. B. Texte, 2D-Grafiken, Animationen, Audioausgaben, Videos oder 3D-Darstellungen zur Anwendung kommen. Die Frage „*Wie* stelle ich etwas dar?“ wird durch den Darstellungskontext beschrieben.

Gerade bei mobilen Anwendungen erschwert die stark begrenzte Darstellungsfläche in Verbindung mit den geringen Auflösungen der Bildschirme die adäquate Präsentation der darzustellenden Information. Die hierdurch notwendige Aufteilung des Inhalts in kleine Präsentationseinheiten führt zudem zu komplexen Navigationsstrukturen.

Die heutigen Anwendungen lösen durch verschiedene Darstellungskontexte entstehende Probleme einfach durch komplett unterschiedliche Versionen für die verschiedenen Ausgabemedien. Dies führt natürlich zu einem entsprechend hohen Entwicklungsaufwand, oft verbunden mit völlig unterschiedlichen Benutzerinterfaces ohne jeglichen Wiedererkennungsfaktor. Zudem wird gerade für leistungsschwächere und insbesondere mobile Geräte häufig nur eine stark eingeschränkte Informationswiedergabe angeboten. Aber auch der umgekehrte Fall einer nur unzureichenden Nutzung der Möglichkeiten leistungsfähiger Systeme lässt sich beobachten.

3.3.1.5 Datenkontext

Die von einer Visualisierungsapplikation darzustellenden Daten hängen im Allgemeinen von den Wünschen und Vorgaben des Benutzers ab und machen eine indi-

viduelle Anpassung und Ausrichtung der verwendeten Daten an den jeweiligen Benutzer erforderlich. In der Regel sind diese Parameter jedoch nicht im Voraus bekannt, sondern werden erst zur Laufzeit definiert. Die benötigten Daten können also nicht vorberechnet werden. Allgemein übliche Algorithmen zur Datengenerierung setzen jedoch häufig auf komplexe Simulationsprozesse, welche zwar qualitativ hochwertige Visualisierungen ermöglichen, jedoch aufgrund des immensen Berechnungsaufwandes bei interaktiven Anwendungen keine Anwendung finden können. Der Datenkontext ist offensichtlich stark von Benutzer- und Situationskontext abhängig.

Die zu generierenden Daten hängen aber auch stark von der aktuell eingesetzten Hardware und den Darstellungsmöglichkeiten ab. So macht es zum Beispiel keinen Sinn, bei einer Datenanfrage eines PDAs zunächst im Datengenerierungsprozess einen komplexen und speicheraufwendigen Datensatz auf einem Server zu berechnen und diesen dann vor der Übertragung durch geeignete Filter wieder zu reduzieren. An dieser Stelle zeigt sich die direkte Abhängigkeit des Datenkontextes vom System- und Darstellungskontext.

Zusammenfassend beschreibt der Datenkontext also die Frage „*Was* stelle ich dar?“.

3.3.2 Agentenbasierte Visualisierungskontrolle

Um die individuellen Anforderungen sowie die interaktiven Anfragen der Benutzer einer Visualisierungsanwendung zu erfüllen und sich an unterschiedliche Hardwarekonfigurationen und sich dynamisch ändernde Systemlasten anpassen zu können, wurde im Rahmen dieser Arbeit eine agentenbasierte Steuereinheit mit einer adäquaten, komponentenorientierten Visualisierungssystem-Architektur kombiniert. Hierdurch wird die dynamische und automatisierte Überwachung und Steuerung aller gewünschten Systembestandteile während der Laufzeit ermöglicht.

Konzeptionell sind hierbei mehrere Ansätze denkbar. Zum einen könnte die Überwachung und Steuerung natürlich von einem einzigen Agent durchgeführt werden, welcher unter Beachtung der Benutzerwünsche die vorliegende Hardwarekonfiguration und -auslastung sowie das Interaktionsverhalten des Benutzers analysiert und durch entsprechende Eingriffe in der Visualisierungspipeline geeignet reagiert. Dieser Ansatz macht jedoch durch sein monolithisches Design für jede Applikation (oder zumindest Applikationsart) jeweils die Implementierung eines speziellen neuen Agenten nötig. Alternativ könnte man den Agenten natürlich auch kontinuierlich mit jeder Anwendung erweitern – die Folge wäre jedoch ein immer größer und entsprechend unübersichtlich werdender Programmcode, so dass diese Umsetzung bereits nach kurzer Zeit kaum noch handhabbar und sinn-

voll erweiterbar wäre. Auch würde die notwendige Analyse des Gesamtsystems zusammen mit der Abarbeitung eines umfangreichen Regelwerkes einem der eigentlichen Ziele, nämlich einer unabhängig von der zur Verfügung stehenden Plattform möglichst guten Performance, aufgrund des entstehenden hohen Overheads widersprechen. Auch wegen der Tatsache, dass moderne Visualisierungsanwendungen aufgrund immer größer werdender Datenmengen oder der benötigten Rechenleistung häufig parallelisiert werden, wurde der monolithische Ansatz nicht weiter verfolgt.

Es liegt damit nahe, statt eines einzelnen Agenten ein Multi-Agenten-System für die Steuerung der Visualisierungsapplikationen einzusetzen, um die komplexen Aufgaben auf einfachere Probleme zu reduzieren. Hierdurch lassen sich nicht nur die beschriebenen Nachteile des monolithischen Ansatzes vermeiden, sondern auch der komponentenorientierte Umsetzungsgedanke völlig analog zu den einzelnen Bausteinen der Visualisierungspipeline anwenden. Um die Multi-Agenten-Technologie einsetzen zu können, müssen die betrachteten Anwendungen drei zentrale Kriterien erfüllen: natürliche Verteilung, dynamische Welten und komplexe Interaktionen. Die hier vorliegenden Szenarien basieren alle auf der Visualisierungspipeline, bei welcher der komplette Visualisierungsprozess in eindeutige Abschnitte aufgeteilt ist. Zudem – vor allem im Falle einer dreidimensionalen Umgebung – liegen äußerst dynamische Welten mit vielfältigen Interaktionsmöglichkeiten durch den Benutzer vor. Daher ist es offensichtlich, dass Visualisierungsaufgaben alle drei Kriterien zur Anwendung der Multi-Agenten-Technologie erfüllen.

Um ein größtmögliches Maß an Flexibilität zu garantieren, muss für jedes zu steuernde Modul bzw. für jede zu steuernde Modulgruppe einer Visualisierungsanwendung ein entsprechender Agent integriert werden. Würde man jedoch lediglich einzelne Agenten ohne Zusammenspiel integrieren, könnte man in der Regel keine Verbesserungen erreichen, im Gegenteil: ein oszillierendes Verhalten durch die völlig unabhängig regelnden Agenten wäre die Folge. Daher müssen sich die Agenten gegenseitig mittels des Austausches geeigneter Informationen entsprechend abstimmen. Beim Einsatz gleichberechtigter Agenten für die Steuerung der Visualisierungsbausteine ist der durch die Kommunikation entstehende Overhead jedoch relativ hoch (jeder Agent muss sich mit allen anderen Agenten im System verständigen), wodurch ähnlich dem monolithischen Ansatz eine unerwünschte Verlangsamung des Visualisierungsprozesses herbeigeführt würde.

Aus diesem Grund kommen im Rahmen dieser Arbeit zwei Typen von Agenten zum Einsatz: einfache reaktive Agenten mit nur lokalem Wissen über die zugehörige Modulfunktionalität sowie ein oder mehrere steuernde / kontrollierende proaktive Agenten mit globalem Wissen und „Kontakt“ zum Benutzer. Ein proaktiver Agent besitzt die Fähigkeit zu zielgerichtetem Handeln und zur Lösung komplexer

Probleme, ist jedoch im Allgemeinen durch die Verwendung komplexer Berechnungsmethoden nicht echtzeitfähig. Reaktive Agenten dagegen können in der Regel nur einfache Aufgaben erfüllen und lassen sich daher gut in dynamische Umgebungen integrieren. In einem Visualisierungssystem bearbeitet ein Modul oder eine Gruppe von Modulen immer die gleiche Aufgabe, so dass die Optimalitätsbedingungen (Framerate, Qualität usw.) durch einfache, statische Regeln beschrieben werden können. Daher wurde den Modulen bzw. den Modulgruppen jeweils ein reaktiver Agent zugeordnet, der die Modulparameter nach Vorgabe steuern kann. Dieser Agent besitzt nur Wissen im Kontext des zugehörigen Moduls; sein Aufbau erinnert an einen Regelkreis.

Die Überwachung der Optimierung sowie die Auswertung und Verarbeitung der von den Modulen gelieferten Informationen übernehmen ein oder mehrere proaktive Agenten. Basierend auf dem Gesamtwissen über die Bestandteile des Systems und den benutzerspezifischen Anforderungen reagieren diese automatisch auf Änderungen in ihrer Umgebung (z. B. auf sich ändernde Systemlasten), indem sie den untergeordneten reaktiven Agenten geeignete Parametervorgaben machen bzw. Justierungsbefehle erteilen.

Im Rahmen dieser Arbeit wurde eine nachrichtenbasierte Kommunikation (vergleiche Kapitel 2.2.4.2) umgesetzt. Die zugrunde liegende Agentenkommunikationssprache basiert auf den Agenten-Kommunikationsstandards KQML (vergleiche Kapitel 2.2.4.3) und KIF (vergleiche Kapitel 2.2.4.4).

Dabei wurden die folgenden Performatives eingesetzt (in der Aufstellung steht *PA* für Proaktiver Agent und *RA* für Reaktiver Agent):

`ask-one`	*PA* fragt *RA* nach dessen aktuellen Einstellungen
`broadcast`	*PA* sendet Befehl an alle *RA*
`tell`	*PA* / *RA* sendet Befehl / Information an einen *RA* / *PA*
`reply`	*PA* / *RA* beantwortet vorangegangene Nachricht eines *RA* / *PA*

Der übertragene Inhalt hängt von der Art der Performative ab. Zum Beispiel kann der steuernde Agent den für den Level of Detail der Geometrie zuständigen Agenten anweisen, den Detaillierungsgrad zu senken oder zurück auf die höchste Auflösung zu schalten. Um die Gesamtperformance so wenig wie möglich negativ zu beeinflussen, ist der durch die Kommunikation entstehende Overhead so gering wie möglich gehalten. Daher wird bei den meisten Befehlen auf die Aufforderung nach einem Reply verzichtet. Nur bei einer Initialisierung, zur Übertragung einer angeforderten Information und beim Auftreten eines Problems (z. B. bei Erreichen der niedrigsten einstellbaren Texturauflösung) sendet der reaktive Agent dem steuernden Agenten einen entsprechenden Hinweis unter Verwendung der `tell`-Performative.

Die einzelnen Kommandos und Antworten sind dabei entsprechend der KQML-Syntax wie folgt aufgebaut:

```
(performative
     :sender nameA
     :receiver nameB
     :reply-with id
     :language VRVAgentLanguage
     :content (command )
```

Abbildung 3-12 greift das Beispiel der Steuerung des für den Level of Detail der Geometrie zuständigen reaktiven Agenten nochmals auf:

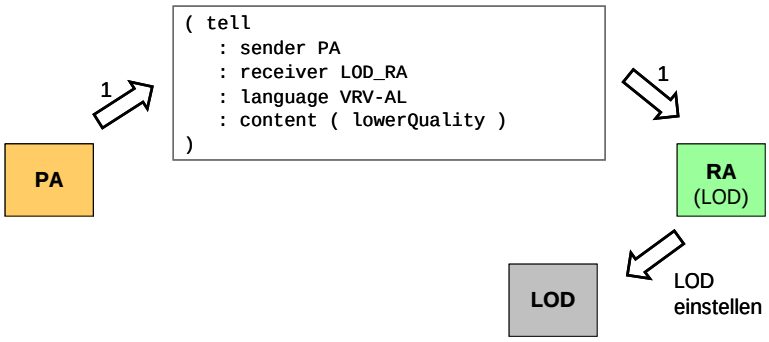

Abbildung 3-12: Steuerung der Qualität und Interaktivität über den LOD.

Sobald der steuernde Agent feststellt, dass die Benutzeranforderungen nicht mehr erfüllt sind, unterrichtet er die verantwortlichen reaktiven Agenten über die gewünschten Optimierungsaktionen. Im vorliegenden Beispiel ist die Renderinggeschwindigkeit zu niedrig - daher teilt der steuernde Agent dem LOD-Agenten mit, dass dieser die Qualität der Szene reduzieren soll. Entsprechend diesem Befehl modifiziert der reaktive Agent die entsprechenden Parameter der LOD-Komponente.

Das beschriebene Gesamtkonzept gibt die Abbildung 3-13 wieder, welche das Zusammenspiel des Multi-Agenten-Systems mit der Visualisierungspipeline veranschaulicht.

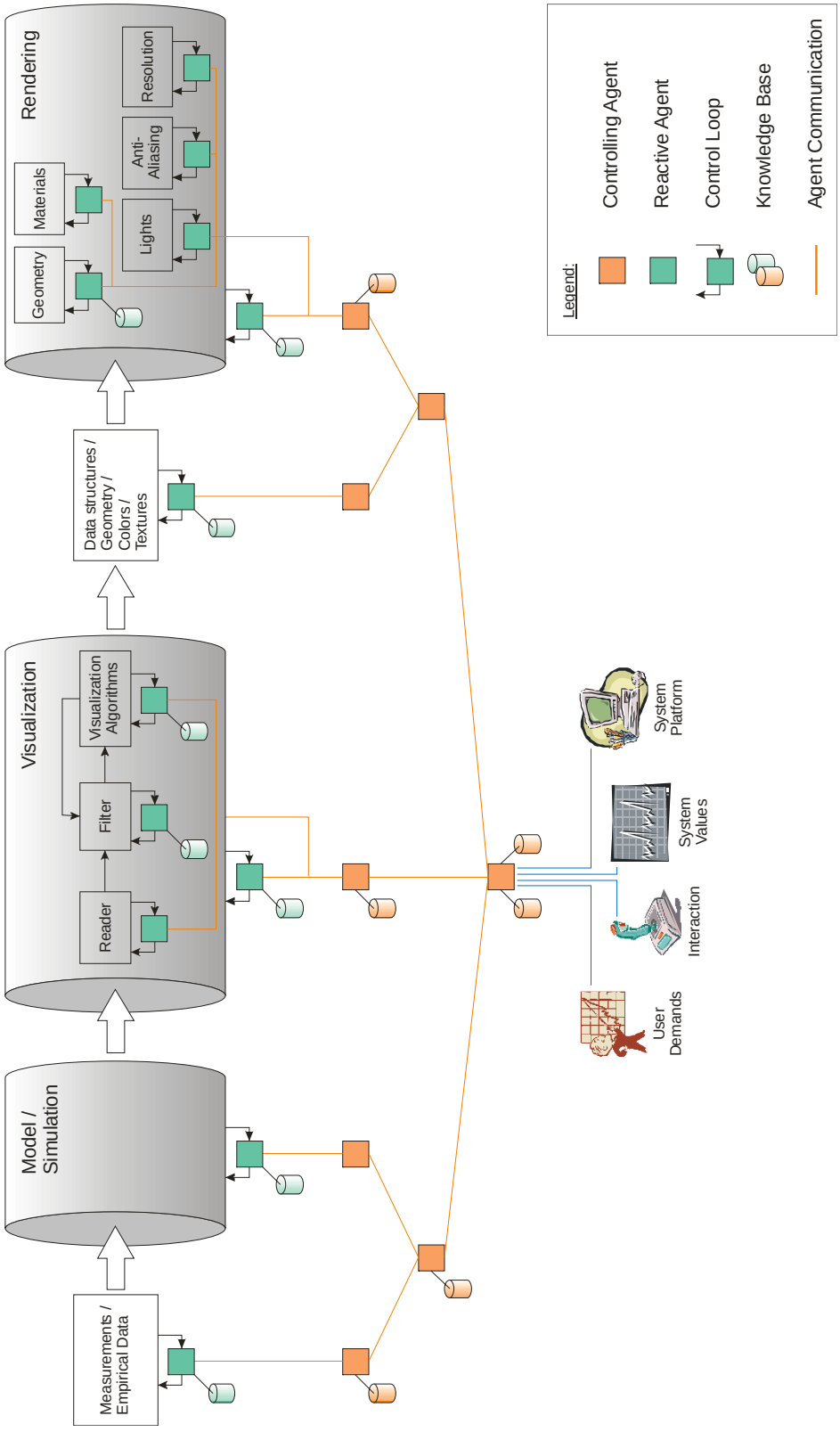

Abbildung 3-13: Agentenüberwachte- und gesteuerte Visualisierungspipeline.

Im oberen Bereich der Abbildung ist die Visualisierungspipeline zu erkennen. Ausgehend von Messdaten oder empirischen Daten werden zunächst in der Modellierungs- bzw. Simulationsstufe der Pipeline die zu visualisierenden Daten generiert und an die Visualisierungsstufe übergeben. Hier werden die Daten dann durch geeignete Filter- und Visualisierungsalgorithmen verarbeitet. Die erzeugten Datenstrukturen zusammen mit Farb- und Materialinformationen sind die Eingabe der Renderstufe. Neben Geometrie und Materialien werden hier Lichtquellen, Aliasing-Parameter und Renderwindow-Auflösung (Fenstergröße, Farbtiefe) gesetzt.

Das zur Überwachung und Steuerung der Pipeline eingesetzte Multi-Agenten-System ist hierarchisch aufgebaut. Den einzelnen Komponenten der Visualisierungspipeline sind die bereits beschriebenen reaktiven Agenten angekoppelt, welche ihr lokales Wissen entweder in lokalen Wissensdatenbanken abgelegt haben oder alle nötigen Informationen direkt vom jeweils übergeordneten steuernden Agenten erhalten. Seine Ziele erhält das Agentensystem durch die Benutzervorgaben, die Interaktion des Benutzers mit der Visualisierung, die Eigenschaften der verwendeten Plattform sowie die zugehörigen dynamischen Systemparameter.

Bereits am Anfang der Pipeline kann ein entsprechender Eingriff erfolgen, nämlich durch geeignete Vorauswahl der eingehenden Daten oder durch entsprechende Steuerung der Messmaschine. Hierdurch kann beispielsweise auf vorab bekannte Parameter aus Benutzer- und Darstellungskontext reagiert werden. Je nach Situation des Benutzers können für diesen andere Daten interessant sein, so dass ebenfalls ein direkter Bezug zum Situationskontext besteht.

Im Bereich der Modellierung und Simulation existieren häufig für den gleichen Anwendungszweck verschieden ausgeprägte Algorithmen, die sich z. B. in Qualität, Geschwindigkeit und Speicherplatzbedarf stark unterscheiden können. Entsprechend der zur Verfügung stehenden Systemperformance (System- und Interaktionskontext) sowie den Vorgaben des Benutzers (Benutzerkontext, insbesondere Vorgaben zu Zeit und Qualität) kann der Agent unter den verschiedenen Algorithmen für die aktuelle Situation die passendste Komponente auswählen und die zugehörigen Parameterwerte setzen. Aber auch der Austausch üblicher Simulationsalgorithmen durch agentengestützte Alternativen ist denkbar. Der Kontextbezug ist hier analog zu den Messdaten.

Der in der Visualisierungsstufe angesiedelte Reader kann verschiedene, durch die Simulationsstufe vorberechnete Ausprägungen, Auflösungs- und Detaillierungsstufen einlesen. Der ihn steuernde reaktive Agent kann daher eine für den aktuellen Benutzer-, System- oder Darstellungskontext möglichst ideale Datenquelle auswählen und entsprechend bereits an dieser Stelle eine einfache Vorfilterung vornehmen. Filterkomponenten nehmen neben reinen Datenanpassungen auch Umrechnungen und Konvertierungen vor. Ein häufig verwendeter Filter ist beispielsweise ein Decimate-Filter, durch welchen eine Datenreduktion vorgenom-

men werden kann. Visualisierungsalgorithmen, wie der Marching Cubes Algorithmus, berechnen schließlich aus wissenschaftlichen Daten bzw. Modellen die eigentlichen Visualisierungsdaten (beispielsweise Geometriedaten, Farbwerte, Texturkoordinaten usw.). Filter und Visualisierungsalgorithmen können bei Bedarf mehrmals nacheinander ausgeführt werden, z. B. um durch Filterung der Simulationsdaten die Visualisierungsdaten zu beeinflussen. Die Parameter der Filter und Visualisierungsalgorithmen unterliegen dabei der Kontrolle von Agenten, welche durch geeignete Konfigurationen die Operationen optimal auf System und Benutzer abstimmen.

Die Visualisierungsdaten können nun wiederum durch einen reaktiven Agenten beeinflusst werden, der Einfluss auf Geometrie und Material (Farbe und Textur) nehmen kann. Eine Möglichkeit ist die Auswahl von Level of Detail-Strukturen oder deren Erzeugung durch geeignete Filterung der Geometrie- oder der Texturdaten. An dieser Stelle wird insbesondere auf den System- und Interaktionskontext eingegangen.

Vom technischen Gesichtspunkt aus gesehen, wirkt sich ein Eingriff in der Renderingstufe vor allem auf den System- und Interaktionskontext aus. So können in diesem Teil der Pipeline zum Beispiel für Performanceanpassungen Teile der darzustellenden Geometrieteile ausgeblendet, Materialien und Lichtquellen an- bzw. abgeschaltet oder Anti-Aliasing-Verfahren zur Erhöhung der Darstellungsqualität angewendet werden. Ebenfalls im Renderer werden die Anpassungen an die Größe, Auflösung und Möglichkeiten des Ausgabemediums vorgenommen und folglich der Darstellungskontext behandelt. Vom technischen Gesichtspunkt her gesehen, wirkt sich ein Eingriff in der Renderingstufe vor allem auf den System- und Interaktionskontext sowie auf den Darstellungskontext aus. Aber auch viele Benutzervorgaben fließen an dieser Stelle ein, so dass ebenfalls ein direkter Bezug zum Benutzerkontext besteht.

In den folgenden Kapiteln 4 und 5 wird der gezeigte Ansatz zur Überwachung und Steuerung der Visualisierungspipeline durch Agenten anhand aussagekräftiger Demonstratoren verdeutlicht und evaluiert. Die Abschnitte 4.1 und 4.2 beschäftigen sich mit der plattformunabhängigen interaktiven Visualisierung und der Anwendung auf medizinischen CT-Datensätzen. Das zentrale Ziel der eingesetzten Agenten liegt hier im Auffinden der aktuell maximal möglichen Performance durch Abwägung der gegenläufigen Hauptparameter Qualität und Interaktivität. Abschnitt 4.3 beschreibt die Entwicklung eines flexiblen, mobilen Informationsvisualisierungssystem im Bereich des Betriebs von Abwasseranlagen, welches durch den Einsatz von Visualisierungsagenten auf Client und Server eine optimale Anpassung an den gerade anfragenden Client ermöglicht. Insbesondere gehen bei diesen Anwendungen der System- und Interaktionskontext sowie der Darstellungskontext ein, in abgeschwächtem Maße auch der Benutzer- und Situations-

kontext. Kapitel 5 konzentriert sich dagegen anhand des Beispiels eines interaktiven Bekleidungskatalogs für eine Virtual Try-On Anwendung vor allem auf die Umsetzung von Daten- und Benutzerkontext. An dieser Stelle wird die übliche zeitaufwendige numerische Simulation durch einen regelbasierten Morphingansatz ersetzt, welcher einen gegebenen Basisbekleidungsdatensatz einer vorgegebenen Größe in ein benutzerindividuelles Bekleidungsstück transformiert.

3.3.3 Komponentenbasierte Umsetzung

Die im Rahmen dieser Arbeit beschriebenen Forschungsarbeiten wurden auf Basis der Java Beans-Architektur umgesetzt. Im Vergleich zu den anderen beschriebenen Komponentenmodellen ist hier die Entwicklung von Komponenten relativ einfach, da sich programmiertechnisch eine Bean nur durch die Einhaltung der Java Beans Design Patterns von einer Standard-Java-Klasse unterscheidet und damit auch als gewöhnliche Java-Klassenbibliothek verwendet werden kann. Auch muss eine Bean nicht von einer Basisklasse abgeleitet werden, wodurch eine von anderen Komponenten unabhängige Entwickelung ermöglicht wird. Darüber hinaus ist eine Verwendung als CORBA-Komponente möglich, sodass man auf diese Weise die Vorteile beider Architekturen nutzen kann.

Basierend auf Java Beans wurde das MACVIS-System (Multi-Agent and Component-based Visualization System) entwickelt, welches erstmals die beiden Hauptnachteile existierender Ansätze

- enge Kopplung zwischen Modul und Visualisierungssystem
- manuelle Anpassung der Modul-Parameter und -Implementierung an die zur Verfügung stehende Hardware und die freien Ressourcen

beseitigt und eine neue Qualität bei der Entwicklung von Visualisierungsanwendungen ermöglicht. Im Rahmen der Entwicklung von MACVIS entstanden das integrierte Component-based Visual Prototyping System CVP, das SciViz-Framework, welches Algorithmen zur Visualisierung als Java Beans-Komponenten bereitstellt, sowie eine Vielzahl von Agenten-Komponenten, die die Komponenten der aufgebauten Visualisierungspipeline überwachen und steuern.

Mittels des CVP-Systems können Anwendungen visuell aus bereits existierenden Java Beans-Komponenten ohne Programmierkenntnisse seitens des Benutzers erstellt werden. An die Komponenten werden außer der Konformität mit den Java Beans Design Pattern keine weiteren Anforderungen gestellt. Insbesondere heißt dies auch, dass keine Basisklassen existieren, von denen ein Baustein abgeleitet werden muss. Somit können die einzelnen Komponenten einer Applikation vollkommen unabhängig voneinander entwickelt und in verschiedenen Entwicklungsumgebungen und Applikationen verwendet werden, die den Java Beans-Anforderungen gerecht werden. Die so vorgefertigten Java Beans-Komponenten

werden zur Laufzeit von dem lokalen Rechner oder über eine Netzwerkverbindung durch Angabe einer URL von einem Server zunächst geladen. Anschließend werden die gewünschten Anwendungen rein visuell durch das Einfügen der benötigten Bausteine mittels Drag & Drop und die Verbindung der In- und Output-Ports durch Connections zur Festlegung des Datenflusses zusammengesetzt. Der Datenfluss selbst ist sofort nach dem Verbinden der Komponenten aktiv, der Benutzer erhält somit ein direktes Feedback über den Ablauf seiner Applikation. Tritt in einer Komponente eine interne Zustandsänderung auf, die nach außen kommuniziert werden soll (Events), so wird diese Zustandsänderung von dem zugeordneten Output-Port an den über die Connection verbundenen Input-Port weitergeleitet.

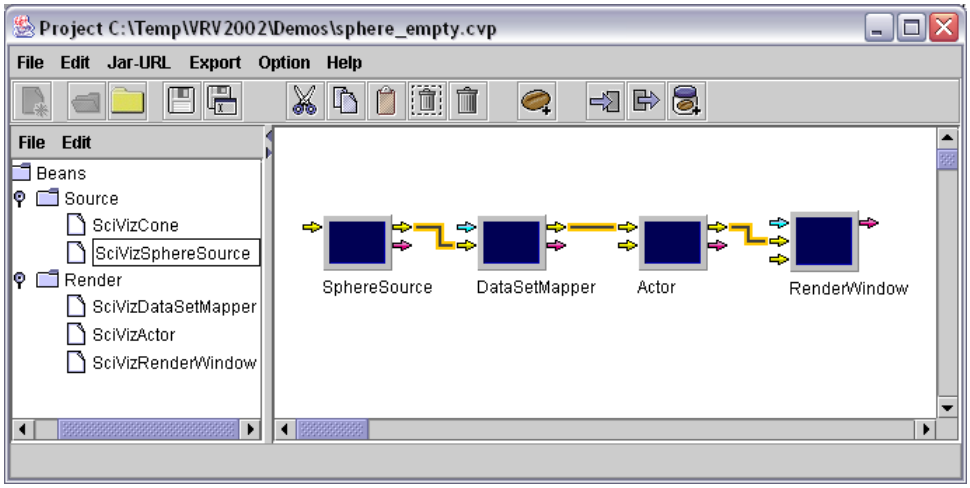

Abbildung 3-14: CVP – Visual Prototyping einer Kugel-Visualisierung.

Die generierte Anwendung kann zur Laufzeit jederzeit durch Hinzufügen oder Löschen von Komponenten und Verbindungen sowie durch Änderung deren Attribute und Einstellungen mit Hilfe eines automatisch erzeugten Customizing-Dialogfensters manipuliert werden. Nach Zusammensetzen und Testen einer Applikation oder Gesamtfunktionalität kann sich der Benutzer hieraus vom System automatisch eine neue Java Beans-konforme Integrationskomponente erzeugen lassen; er muss hierzu lediglich die Input- und Output-Ports des neuen Bausteins definieren.

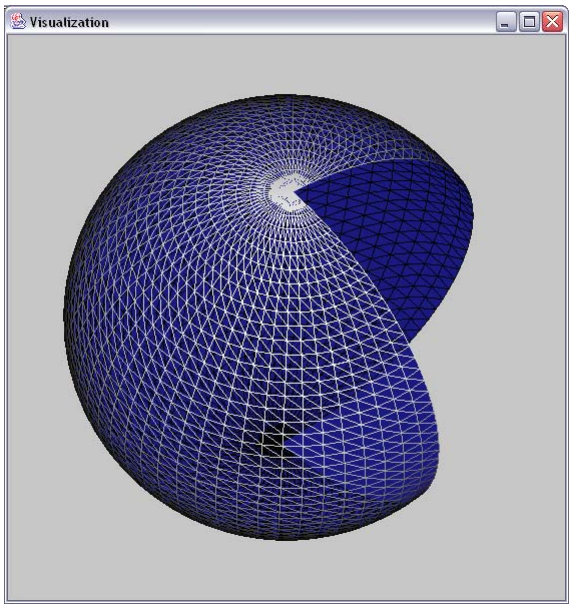

Abbildung 3-15: Visualisierung einer Kugel mit per Customizing editierten Öffnungswinkeln.

Wie bereits erwähnt, sind in diesem Ansatz Visualisierungssystem und Komponenten voneinander unabhängig. Voraussetzung ist dabei, dass eine Komponente analysiert und alle Informationen über sie gesammelt werden können sowie die gewonnene Information ausreicht, um die Kommunikation zwischen Komponenten zu ermöglichen. Basierend auf der Analyse werden die Properties, Methoden und Events jeweils auf spezielle In- und Output-Ports abgebildet, die als Kommunikationsendpunkte dienen.

Neben den Visualisierungsbausteinen sind auch die Agenten unter Beachtung der Java Beans Design Pattern als Komponenten umgesetzt und fügen sich somit nahtlos in die beschriebene Struktur ein. Seitens des Benutzers ist kein Unterschied in der Handhabung zwischen den Visualisierungs- und Agentenbausteinen erkennbar, so dass sich die Vorteile der Komponentenorientierung ohne Einschränkungen nutzen lassen.

4

MACVIS – System- und Darstellungskontext

Das Kürzel MACVIS steht für „Multi Agent- and Component-based Visualization“ und beschreibt die Überwachung und Steuerung der Visualisierungspipeline durch ein Multi-Agenten-System. Das zentrale Ziel der eingesetzten Agenten liegt hier im Auffinden der aktuell maximal möglichen Performance durch Abwägung der gegenläufigen Hauptparameter Qualität und Interaktivität.

Die Abschnitte 4.1 und 4.2 beschäftigen sich mit der plattformunabhängigen interaktiven Visualisierung und deren Anwendung auf medizinische CT-Datensätze. Der beschriebene Ansatz berücksichtigt dabei die Fähigkeiten der eingesetzten Plattform sowie sich dynamisch ändernde Systemwerte, adressiert also vor allem den System- und Interaktionskontext.

Der Gedanke der plattformunabhängigen interaktiven Visualisierung wird in Abschnitt 4.3 auf mobile Informationssysteme ausgeweitet. Hier ist neben den Performanceaspekten vor allem die Art des Ausgabemediums, d. h. der Darstellungskontext, ein entscheidender Faktor. Entsprechend wird bei diesem Demonstrator die darzustellende Information abhängig von der Art, Größe und Auflösung des aktuell verwendeten Gerätes verschieden aufbereitet.

Die innovativen Ideen des MACVIS-Ansatzes sind Inhalt mehrerer Veröffentlichungen ([6],[23],[24],[45]-[48]) auf nationalen und internationalen Konferenzen sowie in Zeitschriften.

4.1 Plattformunabhänige interaktive Visualisierung

Die Anpassung und Optimierung einer Visualisierungsapplikation an eine andere Plattform erweist sich in der Praxis durch die Vielzahl der Parameter und deren gegenseitige Beeinflussung als ein sehr komplexes Problem. Selbst im statischen Fall, also dem Systemkontext, können Experten mit ihrem Wissen und Überblick über das Gesamtsystem diese Problematik nur in begrenztem Umfang lösen. Zusammen mit den einhergehenden, dynamisch wechselnden Bedingungen (d. h. dem Systemkontext), beispielsweise verursacht durch sich ändernde Systemlasten, stellen dies ideale Voraussetzungen für den Einsatz von Agentensystemen zur Steuerung der Visualisierungspipeline dar.

Die in den Abschnitten 4.1 und 4.2 aufgeführten Forschungsarbeiten sind ein wesentlicher Bestandteil des vom Bundesministerium für Bildung und Forschung (BMBF) unter dem Förderkennzeichen 01IW8126 geförderten Projektes *VRV*.

4.1.1 Intelligente Steuerung der Visualisierungspipeline

Das im Rahmen dieser Arbeit entwickelte MACVIS-System kombiniert eine agentenbasierte Steuereinheit mit einer adäquaten, komponentenorientierten Visualisierungssystem-Architektur und ermöglicht somit die Überwachung und Steuerung aller gewünschten Systembestandteile während der Laufzeit.

Die Abwägung der gegenläufigen Hauptparameter Qualität und Interaktivität leistet dabei ein Multi-Agenten-System, welches den zentralen Komponenten der Visualisierungspipeline jeweils einen reaktiven Agenten mit nur lokalem Wissen im Kontext des zugehörigen Moduls zuordnet. Die Überwachung der einzelnen Optimierungen übernimmt ein proaktiver Agent mit Wissen über die gesamte aktuelle Systemumgebung. Der Agent verfolgt dabei das Ziel eine möglichst gute Darstellungsperformance unter Berücksichtigung der Qualitätsanforderungen und Interaktionen des Benutzers zu erreichen. Er reagiert hierzu selbständig auf Änderungen in seiner Umgebung durch Steuerung der untergeordneten reaktiven Agenten. Aufgrund seiner Ziele wird der proaktive Agent im Folgenden auch als Performance-Agent bezeichnet.

Neben dem Wissen über die Möglichkeiten und Auswirkungen der Steuerung der einzelnen reaktiven Agenten muss der Performance-Agent natürlich Informationen über die Leistungsfähigkeit der verwendeten Hardware besitzen, um individuelle Ergebnisse für unterschiedliche Ausgangssituationen berechnen zu können. Hierzu hat er Zugriff auf eine Wissensdatenbank, welche nach Hardwaretyp (z. B. Prozessor, Grafikkarte, Speicher) und Bezeichnung (z. B. Intel Pentium 4 2 GHz, NVidia GeForce 2 MX) geordnet, aussagekräftige Benchmarking- und Vergleichswerte zur Verfügung stellt.

Dem Prozessor sind in der Umsetzung vier verschiedene Benchmarkwerte zugeordnet: Dhrystone [MIPS], Whetstone [MFLOPS], Multi-Media Benchmark (Integer [it/s] und Floating Point [it/s]). Dhrystone und Whetstone sind beide synthetische Benchmarks, die einen für typische Programme repräsentativen Mix von Instruktionen beinhalten. Der ursprünglich von Siemens entwickelte Dhrystone Benchmark wird häufig in der Industrie benutzt, um die CPU-Performance zu messen. Der Benchmark verwendet hierzu eine repräsentative Anzahl von hauptsächlich numerischen Operationen. Das Ergebnis kommt durch die Zeitmessung zustande, die das Programm benötigt, um eine feste Anzahl von Operationen auszuführen. Obwohl das Ergebnis nicht immer die tatsächliche Geschwindigkeit eines gesamten Computersystems repräsentiert, kann mit seiner Hilfe die Leistungsfähigkeit verschiedener Prozessoren gut verglichen werden. Der Whetstone Benchmark wird benutzt, um die Geschwindigkeit des Coprozessors, der FPU (Floating Point Unit), zu messen. Das Ergebnis kommt durch die Zeitmessung zustande, die das Programm benötigt um eine feste Anzahl Gleitkommaberechnungen auszuführen. Beim CPU Multi-Media Benchmark werden intensive Grafikberechnungen durchgeführt, wobei spezielle Prozessorfeatures mit verwendet werden.

Die Leistungsfähigkeit der Grafikkarte bezüglich dreidimensionaler Visualisierung wurde im Rahmen der Arbeit mit dem 3D Mark Benchmark [38] gemessen, womit sich die Leistungsfähigkeit verschiedener Architekturen gut vergleichen lässt. Zusätzlich werden die beiden booleschen Parameter „Beleuchtung in Hardware“ und „Texturierung in Hardware“ verwendet, da bei einer hardwarebasierten Umsetzung z. B. das Abschalten der Textur im vorliegenden Zusammenhang keine nennenswerten Vorteile liefert.

Tabelle 4-1 zeigt einen Ausschnitt aus der Datenbank mit einigen beispielhaften Werten. Dabei steht beim Typ die Abkürzung P für Prozessor, G für Grafikkarte und M für Speicherausbau. Die Werte der Wissensdatenbank sind dabei aus mehreren Durchläufen gemittelt und auf volle 10er gerundet.

Typ	**Bezeichnung**	**Benchmark / Wert**
P	Intel Pentium II 400 MHz	1080, 540, 1450, 580
P	Intel Celeron 600 MHz	1620, 800, 3260, 3980
P	Intel Pentium III 1 GHz	2700, 1340, 5430, 6640
P	Intel Pentium 4 2 GHz	3690, 2440, 7880, 9630
P	AMD Athlon XP 1800+	4240, 2120, 8370, 9760
G	NVidia Geforce 3 Ti 500	9140, 1, 1

Typ	Bezeichnung	Benchmark / Wert
G	ATI Radeon 8500	8940, 1, 1
G	Matrox G450	2460, 0, 0
M	512 MB	512
M	1 GB	1024

Tabelle 4-1: Ausschnitt aus der Benchmark-Datenbank.

Anhand der einzelnen Werte des Systems können nun für verschiedene Plattformen individuelle Performance-Indices berechnet werden, anhand derer beispielsweise die Visualisierungsbausteine initialisiert sowie geeignete LOD-Auflösungen vorberechnet werden können. Für einen „schnellen“ Rechner könnten dies bei einem Ausgangsobjekt mit 100.000 Dreiecken z. B. die fünf LOD-Stufen 10.000, 30.000, 50.000, 75.000 und 100.000 Dreiecke sein, während für einen „langsamen“ Rechner niedrig angesiedeltere Stufen, z. B. 1.000, 5.000, 10.000, 20.000 und 100.000 Dreiecke, bestimmt werden. Der Grund für die unterschiedlichen Abstufungen ist, dass auf einem „schnellen“ Rechner geringe Auflösungen in feiner Abstufung, auf einem langsamen Rechner hohe Auflösungen in feiner Abstimmung nicht sinnvoll sind, da diese nicht oder nur sehr selten aufgerufen würden.

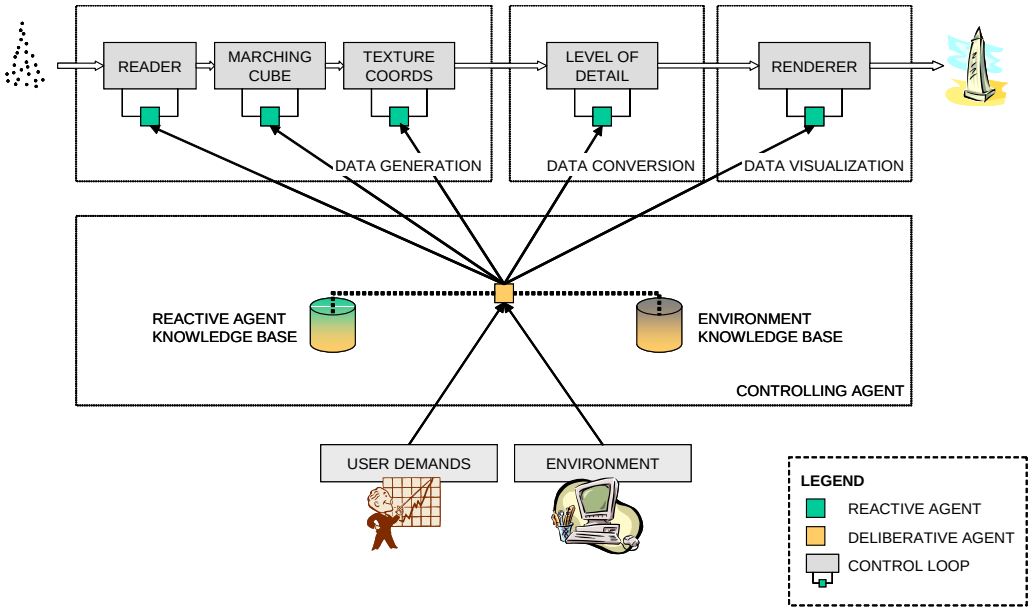

Abbildung 4-1: Steuerung der Visualisierungspipeline durch Agenten.

Abbildung 4-1 zeigt ein Beispiel für eine von Agenten kontrollierte Visualisierungspipeline. Im „Data Generation“-Prozess wird ein CT-Datensatz eingelesen

und nachfolgend der Marching-Cubes-Algorithmus angewandt. Nach der Erzeugung und Zuweisung der Texturkoordinaten werden im „Data Conversion“-Prozess verschiedene, vordefinierte Detaillierungsgrade berechnet. Innerhalb des eigentlichen Datenvisualisierungsprozesses wird die ausgewählte Objektrepräsentation gerendert, d. h. sie wird auf die darunter liegende Hardwareabstraktionsebene abgebildet.

Um die vom Benutzer vorgegebene Renderingqualität und Performance zu erreichen, werden jeder Komponente der Pipeline reaktive Agenten zugeordnet, die vom Performance-Agenten überwacht werden. Beispielhaft soll im Folgenden kurz das Zusammenspiel der Data Conversion-Module, der Data Visualization-Module und dem Performance-Agenten beschrieben werden: Der reaktive Agent des Data Conversion-Moduls kontrolliert die Level of Detail-Komponente durch Änderung der den Level of Detail-Algorithmus, den Detaillierungsgrad und die Genauigkeitsanforderungen betreffenden Parameter. Die Ausgabe des Data Conversion-Moduls ist der effektive Detaillierungsgrad und die Anzahl der zugehörigen Dreiecke. Der reaktive Agent des Datenvisualisierungsmoduls kontrolliert die Rendering-Komponente durch Variieren des Darstellungsmodells, des Schattierungsmodells, des Renderingalgorithmus, der Auflösung und der Beleuchtungs- sowie der Texturing-Parameter. Die Ausgabe des Datenvisualisierungsmoduls ist die aktuelle Framerate bzw. die zugehörige für das Rendering benötigte Zeit. Beide Module erhalten ihre erlaubten Parameterbereiche vom Performance-Agenten, der seinerseits von den Benutzeranforderungen und der aktuellen Systemumgebung beeinflusst wird.

4.1.2 Kommunikation

Im Rahmen dieser Arbeit wurde wie in Kapitel 3.3 beschrieben zwischen Performance-Agent und reaktiven Agenten eine nachrichtenbasierte Kommunikation basierend auf den Agenten-Kommunikationsstandards KQML und KIF umgesetzt.

Im Folgenden soll der Ablauf kurz an einem Beispiel verdeutlicht werden. Die Aufgabenstellung ist die interaktive Visualisierung eines Datensatzes auf einem beliebigen PC. Hierzu kommen bei der Visualisierungspipeline die Module *Load* (Laden von vorberechneten Datensätzen), *Decimate* (Berechnung eines bestimmten LODs) und *Visualize* (Rendern der Szene) zum Einsatz. Das *Visualize*-Modul besteht intern aus den Modulen *Geometry* (Bereitstellung der notwendigen Geometrie), *Texture* (Texturierung des Objektes) und *Light* (Lichtquellen in der Szene). Alle Einzelbausteine erhalten zur Steuerung der Parameter einen reaktiven Agenten zugeordnet, die Gesamtkontrolle übernimmt der Performance-Agent. Das Zusammenspiel ist in Abbildung 4-2 veranschaulicht.

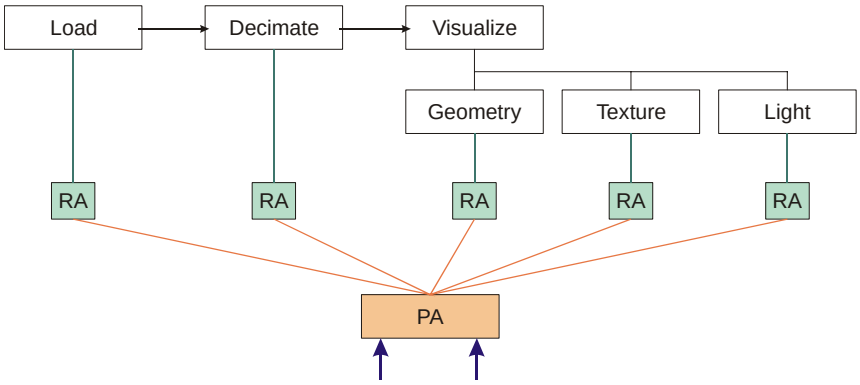

Abbildung 4-2: Prinzipieller Aufbau einer Applikation zur interaktiven Visualisierung.

Der Ablauf nach Start der Applikation gestaltet sich dann wie folgt, wobei insgesamt 10 verschiedene LODs zur Verfügung stehen sollen:

i) Zunächst bestimmt der PA (Performance-Agent) anhand der Leistungsfähigkeit des zur Verfügung stehenden Systems die maximale Anzahl von Dreiecken, die bei einer interaktiven Anwendung gerade noch sinnvoll erscheint.

ii) Anhand der Vorgaben des Benutzers zur gewünschten Qualität der Darstellung bestimmt der PA die untere Grenze für die Anzahl der Dreiecke.

iii) Der PA gibt nun dem Load-RA (Reaktiver Agent) die Anweisung, 10 verschiedene LODs (mit gewissem Mindestabstand bei der Anzahl der Dreiecke) für das in den Schritten i) und ii) bestimmte Intervall zu laden.

iv) Der Load-RA steuert entsprechend das Laden der benötigten LODs und antwortet dem PA mit einer Liste der geladenen Auflösungsstufen.

v) Waren nicht alle 10 benötigten LODs vorberechnet und konnten folglich auch nicht im Schritt iv) geladen werden, so gibt der PA dem Decimate-RA die Anweisung zum Anstoßen der Berechnung der fehlenden LODs.

vi) Der Decimate-RA steuert die Berechnung der fehlenden LODs und informiert den PA sobald diese berechnet sind.

vii) Der PA leitet die Liste der zur Verfügung stehenden LODs an den Geometry-RA weiter.

viii) Der PA misst nun fortlaufend die Interaktivität des Systems anhand der aktuellen Framerate. Ist die Framerate zu niedrig, so weist er beispielsweise den Geometry-RA an, eine oder die nächst niedrigere LOD-Stufe anzuzeigen. Liegt die Framerate deutlich über der vom Benutzer geforderten Framerate, so veranlasst der PA entsprechend die Anzeige einer höheren Stufe.

Diese Abfolge spiegelt sich entsprechend in den einzelnen KQML-Kommandos und -Antworten wider. In Schritt iii) wird vom PA mit einer `tell`-Performative

dem RA eine Liste von Parametern übermittelt, der `content`-Bereich lautet entsprechend `(listof #LODs (listof min max))`. In diesem Fall wird ein Reply angefordert, der RA sendet also ein `reply` mit der Anzahl und der Liste der geladenen LODs als `content`. Bei der eigentlichen Steuerung der Visualisierung sendet der PA ohne Anforderung einer Antwort via `tell` die Nachricht `(lowerQuality)` zur Herabsetzung der Qualität um eine Stufe oder die Nachricht `(lowerQuality (x))`, wobei `x` die benötigte Qualität in Prozent im Vergleich zur aktuell gewählten Qualitätsstufe angibt. Der RA wählt entsprechend den am nächsten liegenden LOD aus. Analog wird die Qualität durch die Verwendung des Befehls `(higherQuality)` bzw. `(higherQuality (x))` wieder heraufgesetzt. Mittels einer `tell`- oder `broadcast`-Performative sowie den Kommandos `(turnOn)` bzw. `(turnOff)` können ein oder alle Agenten im System an- bzw. abgeschaltet werden.

Die folgenden Abbildungen zeigen beispielhaft einen Kommunikationsprozess zwischen dem steuernden proaktiven Agenten und dem reaktiven Agenten, der für die Wahl geeigneter LOD-Stufen verantwortlich ist.

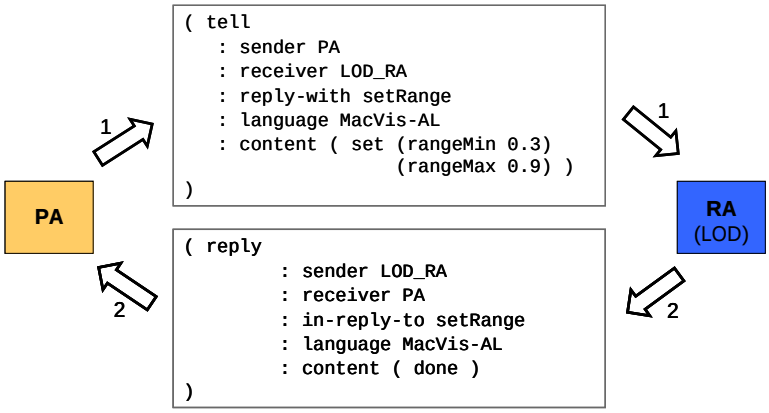

Abbildung 4-3: Initialisierung des LOD-Agenten.

Basierend auf dem Wissen über die gesamte Umgebung, die einbezogenen reaktiven Agenten und die Benutzeranforderungen ermittelt der steuernde Agent die jeweils erlaubten Parameterbereiche und teilt diese in einem Initialisierungsschritt den entsprechenden Agenten mit (Abbildung 4-3). Es sei angemerkt, dass diese Parameter in Fuzzy-Logik interpretiert werden, im Beispiel reicht so die Qualität von gering (0,3) bis sehr hoch (0,9).

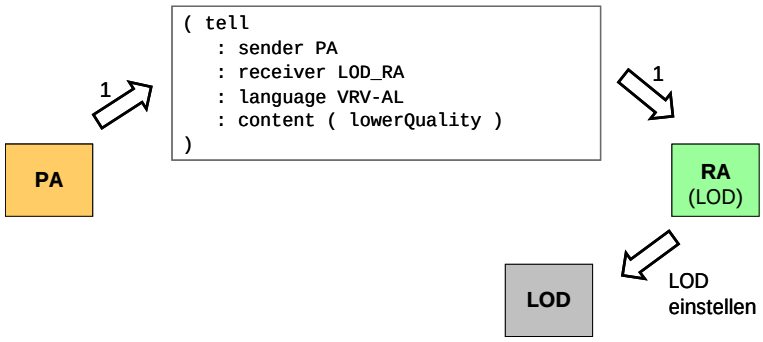

Abbildung 4-4: Steuerung der Qualität und Interaktivität über den LOD.

Sobald der steuernde Agent feststellt, dass die Benutzeranforderungen nicht mehr erfüllt sind, unterrichtet er die verantwortlichen reaktiven Agenten über die gewünschten Optimierungsaktionen. Im vorliegenden Beispiel ist die Renderinggeschwindigkeit zu niedrig, daher teilt der steuernde Agent dem LOD-Agenten mit, dass dieser die Qualität der Szene reduzieren soll (Abbildung 4-4). Entsprechend diesem Befehl modifiziert der reaktive Agent die Parameter der LOD-Komponente.

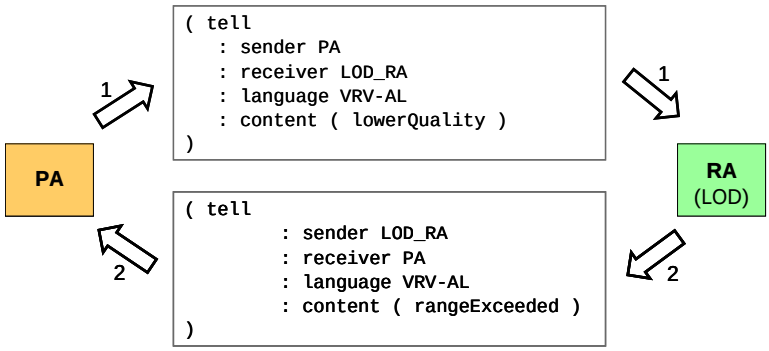

Abbildung 4-5: Fehler- bzw. Ausnahmebehandlung des LOD-Agenten.

Falls diese Befehlsausführung zu Werten außerhalb des erlaubten Parameterbereichs führen würde, teilt der LOD-Agent dieses dem steuernden Agenten mit (Abbildung 4-5).

4.2 Anwendung für CT-Datensätze

Die Aufgabe der Scientific Visualization liegt in der Visualisierung von Daten, die durch Simulationen oder Messungen entstanden sind. In der Regel sind diese Daten nicht kontinuierlich sondern diskret gegeben. Ein Beispiel für solche Datensätze sind die CT (Computer-Tomografie)-Daten in der Medizin. Bei dieser Tech-

nik werden regelmäßige Schnitte durch den zu untersuchenden Bereich erzeugt. Dabei beschreibt jeder Datenpunkt (x_i,y_j,z_k,G_{ijk}) die Gewebedichte G_{ijk} am Punkt $(x_i,y_j,z_k) \in \Re^3$. Diese Punkte liegen auf einem regelmäßigen Gitter im Raum, typische Auflösungswerte für x und y sind 64, 128 und 256. Die die Gewebedichte symbolisierenden Grauwerte sind häufig mit 8 Bit Auflösung digitalisiert. Durch das riesige Datenaufkommen und die großen Diskrepanzen zwischen Visualisierung und verwendeter Hardware eignen sich CT-Datensätze besonders gut, um die Anwendung der agentenbasierten Visualisierungskontrolle zu demonstrieren.

Zu Beginn der Betrachtungen wird zunächst der häufig für die Visualisierung von CT-Daten eingesetzte Marching Cubes-Algorithmus zur Berechnung von Konturflächen beschrieben, bevor dieser dann als Visualisierungsalgorithmus zum Einsatz kommt. Die Anzahl der durch diesen Algorithmus erzeugten Dreiecke ist sehr groß, eine interessante Möglichkeit zur automatischen Visualisierungskontrolle liegt daher in der dynamischen Auswahl entsprechend angepasster Auflösungsstufen. Man spricht hier von Level of Detail-Verfahren, in diese Thematik soll daher im Folgenden ebenfalls kurz eingeführt werden.

4.2.1 Der Marching Cubes-Algorithmus

Der Marching Cubes-Algorithmus [76] erweitert die Idee des im Zweidimensionalen arbeitenden Marching Squares-Algorithmus zur Berechnung von Konturlinien auf den dreidimensionalen Fall. Daher wird im Folgenden zunächst auch der Marching Squares-Algorithmus beschrieben. Kontur- oder Isolinien sind Linien gleichen Funktionswertes, welche also für eine Funktion f alle Punkte mit $f = C$ darstellen. Bei diskreten Messwerten wie im Falle von CT-Datensätzen wird dieser Wert C jedoch nicht immer an den Gitterpunkten angenommen, es gibt dafür aber Nachbarpunkte, die erwarten lassen, dass der Wert C auf der Kante zwischen den Punkten angenommen wird. Es wird dabei angenommen, dass ein Wert nur einmal angenommen wird und dass das Gitter so fein gewählt ist, dass die Konturlinien innerhalb der Zellen annähernd linear verlaufen bzw. durch lineare Interpolation hinreichend gut approximiert werden.

Im ersten Schritt wird nun eine Kante gesucht, auf der der Wert C angenommen wird. Gibt es eine solche Kante nicht, so gibt es auch keine Konturlinie $f = C$ in dem untersuchten Datensatz. Nach den oben getroffenen Annahmen muss eine Konturlinie, die eine Zelle betritt, diese auch wieder verlassen, d. h. auf mindestens einer der anderen drei Kanten dieser Zelle muss ebenfalls der Wert C angenommen werden. Diese „Ausgangskante“ ist gleichzeitig wieder „Eingangskante“ für die benachbarte Zelle, die nun entsprechend abgearbeitet wird. Wird eine Randkante erreicht, so verlässt die Konturlinie an dieser Stelle den Datensatz. Die Vorstellung des „Wanderns von einer Zelle zur nächsten“ führte dann auch zur Begriffsbildung „Marching Squares“.

Die folgende Abbildung zeigt ein 5x5-Gitter mit den Konturlinien für *C*=5 (rot), *C*=3 (blau) und *C*=2 (grün). Dabei wird am Fall *C*=2 deutlich, dass eine Konturlinie auch aus mehreren, unzusammenhängenden Teilen und im Extremfall stückweise nur aus einer Linie bestehen kann.

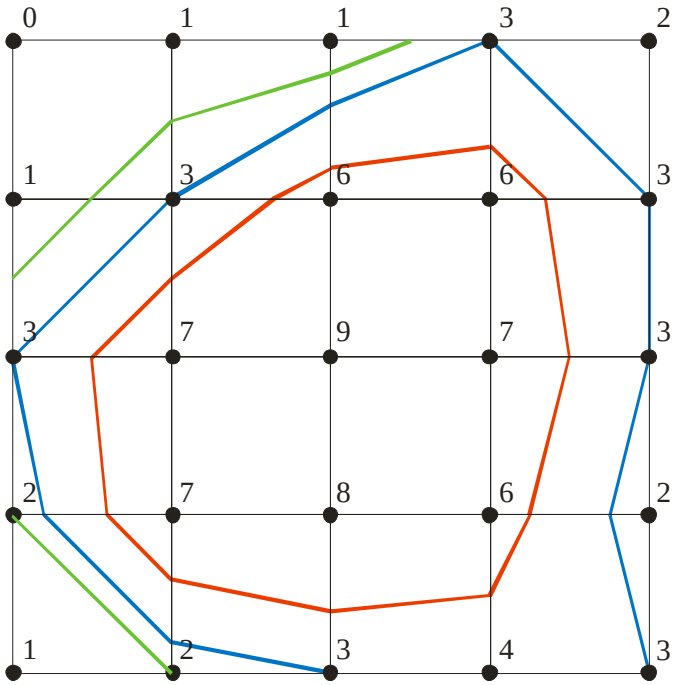

Abbildung 4-6: Konturlinien in einem Datensatz auf einem 5x5-Gitter.

Da die Konturlinien nach Annahme innerhalb der einzelnen Zellen linear verlaufen, ist die Anzahl der Möglichkeiten, wie eine Konturlinie eine Zelle schneiden kann, begrenzt. Hierzu betrachtet man lediglich, ob der Wert in einem Gitterpunkt größer gleich oder kleiner als der Wert *C* ist. Ist der Wert im betrachteten Gitterpunkt größer oder gleich *C*, so codiert man den Gitterpunkt mit einer 1, ansonsten mit einer 0. Im betrachteten zweidimensionalen Fall existieren lediglich $2^4 = 16$ Möglichkeiten, da eine Zelle vier Gitterpunkte besitzt, welche jeweils zwei Zustände annehmen können (siehe Abbildung 4-7). In der Abbildung markiert ein ausgefüllter Gitterpunkt eine 1, ein unausgefüllter Punkt eine 0.

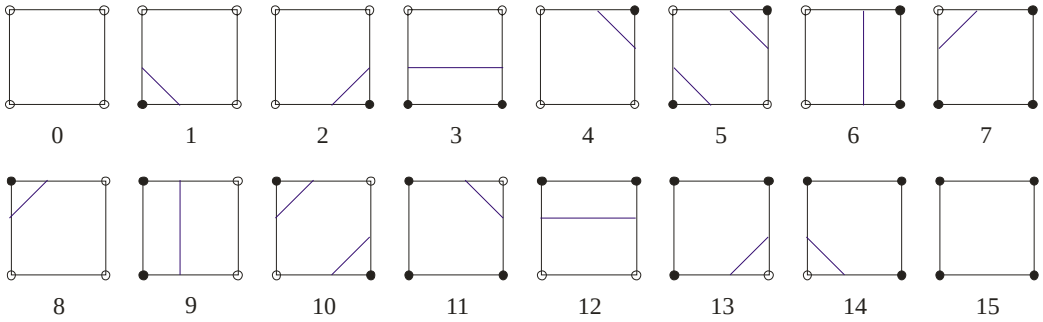

Abbildung 4-7: 16 Schnittarten beim Marching Squares-Algorithmus.

Bezeichnet man die Eckpunkte eine Zelle unten links beginnend gegen den Uhrzeigersinn mit V_1 bis V_4, so kann der jeweils dargestellte Zustand durch den Bitvektor (V_4, V_3, V_2, V_1) codiert werden (in obiger Abbildung ist der dezimale Wert des Bitvektors als Fallnummer angegeben). Entsprechend wird für alle möglichen Fälle eine Lookup-Table aufgestellt.

Bei Betrachtung der verschiedenen Zustände fällt auf, dass die Zustände 5 und 10 keinen eindeutigen Verlauf der Konturlinie erlauben, es sind jeweils drei Fälle denkbar (siehe Abbildung 4-8).

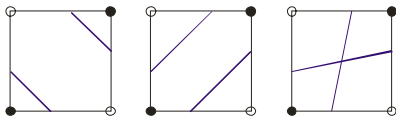

Abbildung 4-8: Nicht eindeutiger Verlauf der Konturlinien (Fall 5).

In den meisten Algorithmen wird ausgeschlossen, dass sich Konturlinien überkreuzen können. Die Entscheidung für einen der beiden anderen Fälle kann man dann z. B. durch eine weitere, entsprechend feinere Unterteilung treffen. Der Einfachheit halber wird oft auch einfach eine der Alternativen ausgewählt und die Art der Entscheidung dann für alle Zellen in identischer Form durchgeführt.

Hat man nun den Zustand einer Zelle bestimmt, so berechnet sich der Schnittpunkt S der Konturlinie mit der durch die Gitterpunkte V_i und V_j definierten Kante über die Konvexkombination:

$$S = (1-\lambda)V_i + \lambda V_j, \quad \text{mit} \quad \lambda = \frac{C - f(V_i)}{f(V_j) - f(V_i)}, \quad (0 \le \lambda \le 1)$$

Zusammenfassend lässt sich der Marching Squares-Algorithmus damit wie folgt aufschreiben:

1) Wähle eine Zelle aus.
2) Bestimme den Zustand ihrer Eckpunkte und den sich dadurch ergebenden Bitvektor.
3) Entnehme aus der Lookup-Table die Art des Verlaufes der Konturlinie in der betrachteten Zelle.
4) Berechne den bzw. die Schnittpunkte der Konturlinie mit den entsprechenden Kanten der Zelle.
5) Gehe zur nächsten Zelle bis alle Zelle durchlaufen wurden.

Die Vorteile des Marching Squares-Algorithmus liegen vor allem in der relativ einfachen Implementierbarkeit und der Tatsache, dass er alle Konturlinien im Datensatz findet.

Der Marching Cubes-Algorithmus erweitert die Idee der Marching Squares nun auf den dreidimensionalen Fall. Die Grundlage bildet hier ein Datensatz (x_i,y_j,z_k,G_{ijk}) mit skalaren Werten G_{ijk} an den Gitterpunkten (x_i,y_i,z_i) eines rechtwinkligen dreidimensionalen Gitters. Entsprechend dem zweidimensionalen Fall sollen für eine Funktion f Konturflächen für $f = C$ berechnet werden. Die einzelnen Zellen sind nun Voxel mit 8 Gitterpunkten, welche analog dem Marching Squares-Ansatz mit 0 oder 1 klassifiziert werden. Damit gibt es im dreidimensionalen Fall insgesamt $2^8 = 256$ Möglichkeiten, wie die Konturfläche eine Zelle schneiden kann. Jeweils zwei Fälle erzeugen dabei jedoch das gleiche Dreieck, diese Fälle bezeichnet man als komplementär. Bei den verbleibenden 128 Möglichkeiten sind dann zusätzlich noch jeweils 8 Fälle bis auf Rotation identisch, so dass sich die tatsächlich durch den Algorithmus zu betrachtenden Möglichkeiten auf 15 reduzieren lassen. Wie schon im zweidimensionalen Fall ist der Verlauf der Konturflächen in einigen Fällen nicht eindeutig, die dort vorgestellte einfache Lösung der durchgängigen Entscheidung für eine bestimmte Alternative lässt sich jedoch nicht ins Dreidimensionale übertragen. In der Literatur existieren zur Lösung dieses Problems eine Vielzahl von Ansätzen, deren Betrachtung jedoch nicht Teil dieser Arbeit ist (ein geeigneter Ansatz findet sich z. B. in [84]). Nach dem Durchlaufen aller Voxel des Gitters liegt ein Dreiecksnetz vor, welches die Konturfläche approximiert. Je nach Anwendung kann dieses dann noch zu Triangle Strips verbunden, mittels Decimation oder Progressive Meshes reduziert und durch die Berechnung gemittelter Normalenvektoren in den Dreiecksecken für die Visualisierung mit Gouraud- oder Phong-Shading vorbereitet werden. Der Marching Cubes-Algorithmus unterliegt einem Patent (United States Patent Number 4,710,876) von General Electric.

Zusammenfassend lässt sich der Marching Cubes-Algorithmus wie folgt beschreiben:

1) Wähle eine Zelle aus.
2) Bestimme den Zustand ihrer Eckpunkte und klassifiziere die Zelle entsprechend.
3) Entnehme aus der Klassifizierung die Art des Verlaufes der Konturfläche innerhalb der betrachteten Zelle.
4) Berechne die Schnittpunkte der Konturfläche mit den entsprechenden Kanten der Zelle über eine Konvexkombination. Zusammen mit Schritt 3 ergeben sich so die Dreiecke der Konturfläche in dieser Zelle.
5) Gehe zur nächsten Zelle bis alle Zelle durchlaufen wurden.

4.2.2 Level of Detail

Wie viele andere Verfahren auch, erzeugt der Marching Cubes-Algorithmus je nach Auflösung des Gitters ein sehr detailliertes und komplexes Modell des untersuchten Objektes. Häufig – und gerade bei der interaktiven Visualisierung – reicht zur adäquaten Darstellung auch ein entsprechendes Objekt mit weniger Facetten aus. Ziel der so genannten Level of Detail-Verfahren ist daher die Erzeugung einer Darstellung, die visuell zufrieden stellend ist und aus möglichst wenigen Facetten besteht. Dabei ist das Kriterium „visuell zufrieden stellend“ äußerst dynamisch – so genügen beispielsweise bei einer Interaktion oder bei weit entfernten Objekten deutlich geringer aufgelöste Objekte aus als bei einem Objekt im Vordergrund ohne gleichzeitige Interaktion des Betrachters. Man benötigt also jeweils verschiedene Auflösungsstufen der darzustellenden Objekte.

Die einfachste Möglichkeit zur Konstruktion von Objekten verschiedener Auflösungsstufen besteht natürlich in der Konstruktion eines Modells für jeden verwendeten Level of Detail. Diese Vorgehensweise ist jedoch sehr aufwendig, zudem können direkt beim Start der Applikation oder zur Laufzeit keine für die aktuelle Situation gegebenenfalls besser geeigneten Auflösungsstufen berechnet werden. In der Regel werden daher Algorithmen eingesetzt, die ein gegebenes Netz entsprechend automatisch dezimieren können. Die Vorgehensweise ist dabei in der Regel iterativ, d. h. in jedem Schritt wird nur eine Kante oder eine Facette gelöscht. Dies wird im Folgenden am Beispiel des von Hoppe et al. entwickelten Verfahrens zur Vereinfachung und Optimierung von Netzen [55] kurz vorgestellt.

Gesucht ist ein Netz, welches die gegebenen Daten möglichst gut approximiert und gleichzeitig eine möglichst kleine Anzahl von Eckpunkten besitzt. Anders formuliert soll also für ein gegebenes polygonales Netz M_0 und eine Menge von Punkten $X \subset \Re^3$ ein polygonales Netz M konstruiert werden, welches das gleiche

Geschlecht (d. h. Anzahl der Löcher und Henkel) wie M_0 hat, die Punkte der Menge *X* hinreichend gut approximiert und eine möglichst kleine Anzahl von Eckpunkten besitzt. Dieses Optimierungsproblem wird mathematisch durch die Minimierung folgender Zielfunktion *E(K,V)* beschrieben:

$$E(K, V) = E_{dist}(K, V) + E_{rep}(K) + E_{feder}(K, V)$$

Dabei bezeichnet *V* die Menge der Eckpunkte des Netzes *M* und *K* die Menge der Adjazenzbeziehungen (Kanten und Facetten) zwischen den Eckpunkten.

Die Funktion E_{dist} beschreibt eine Distanzfunktion und gibt den Abstand zwischen der Punktmenge *X* und dem zu konstruierenden Netz *M* an:

$$E_{dist}(K, V) = \sum_{i=1}^{n} dist^2(X_i - M)$$

Die Funktion E_{rep} misst die Anzahl der Eckpunkte. Der Parameter c_{rep} stellt dabei eine Gewichtung dar, die zwischen den beiden Forderungen möglichst kleiner Abstand zwischen *M* und *X* auf der einen Seite und möglichst kleiner Anzahl von Eckpunkten auf der anderen Seiten balanciert:

$$E_{rep} = c_{rep}|V|$$

Wäre nun *E(K,V)* nur über E_{dist} und E_{rep} definiert, so könnten in der damit bestimmten optimalen Lösung (lokales Minimum) Flächenstücke in Bereichen entstehen, in denen gar keine Punkte gegeben waren. Daher wird der Term E_{feder} zur Regularisierung der Zielfunktion aufgenommen:

$$E_{feder} = \sum_{(V_j, V_k) \in K} \kappa \|V_j - V_k\|^2$$

E_{feder} summiert über alle Kanten des polygonalen Netzes und sorgt dafür, dass ein gutes lokales Minimum gefunden wird.

Um die Zielfunktion über der Menge aller möglichen Eckpunktkoordinaten und aller möglichen Adjadenzbeziehungen zwischen diesen minimieren zu können, wird die Optimierung in zwei Schritten durchgeführt.

Zuerst wird in einer „inneren Optimierung“ die Topologie festgehalten und eine möglichst gute Position der Eckpunkte berechnet. Diese Optimierung kann man als eine Projektion der Punkte der Menge X auf das Netz und ein lineares Ausgleichsproblem formulieren. Damit ist eine effiziente Lösung der inneren Optimierung möglich.

Für die „äußere Optimierung“ wird bei festgehaltenen Eckpunkten die Topologie so verändert, dass ein Abstieg in der Zielfunktion erreicht wird. Hierzu werden drei elementare Operationen auf Netzen eingeführt, die in Abbildung 4-9 veranschaulicht sind: Edge Collapse, Edge Split und Edge Swap.

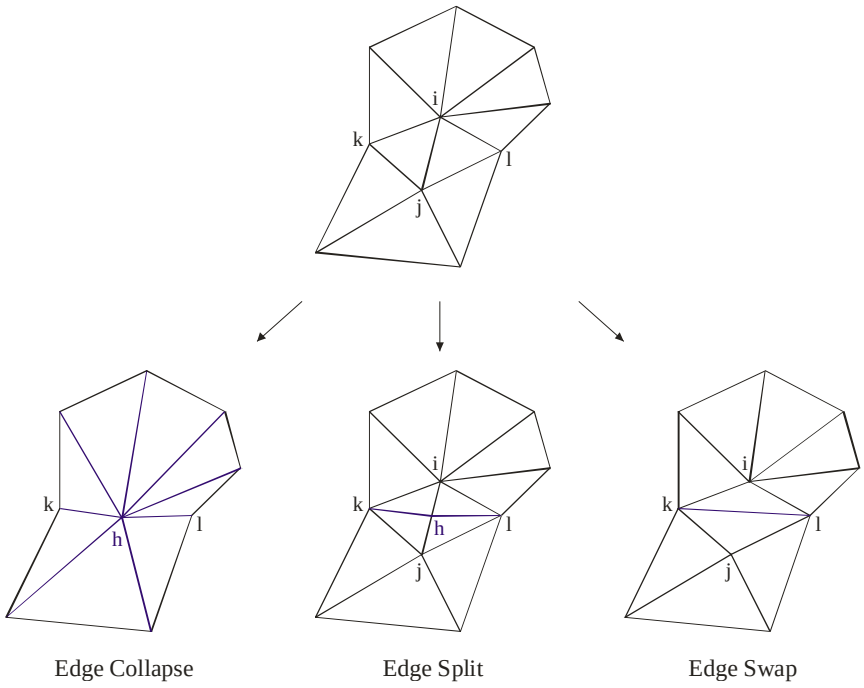

Abbildung 4-9: Elementare Operationen auf Netzen.

Bei einem Edge Collapse werden die beiden Eckpunkte V_i und V_j einer Kante zu einem neuen Eckpunkt V_h verschmolzen. Durch diese Operation wird die Kante (V_i,V_j) und die beiden Dreiecke, die diese Kante als Rand besitzen, aus dem Netz gelöscht. Ein Edge Collapse ist nur dann zulässig, wenn die folgenden vier Kriterien erfüllt sind:

i) für alle zu V_i und V_j benachbarten Eckpunkten V_k ist (V_i,V_j,V_k) eine Facette im Netz;

ii) sind V_i und V_j Randpunkte des Netzes, so ist auch (V_i,V_j) eine Randkante;

iii) im Netz gibt es insgesamt mehr als 4 Eckpunkte, falls sowohl V_i als auch V_j keine Randpunkte ist;

iv) im Netz gibt es insgesamt mehr als 3 Eckpunkte, falls entweder V_i oder V_j ein Randpunkt ist.

Der Edge Split stellt die inverse Operation zum Edge Collapse dar. Hierzu wird auf der Kante (V_i,V_j) ein neuer Eckpunkt $V_h = 0.5(V_i + V_j)$ eingefügt; aus den direkt angrenzenden zwei Dreiecken werden also vier Dreiecke erzeugt.

Ein Edge Swap ersetzt die Kante (V_i,V_j) durch die Kante (V_k,V_l), falls die Punkte analog Abbildung 4-9 (rechts unten) liegen. Der Edge Swap ist nur dann zulässig, wenn die neu entstandene Kante nicht schon im Netz enthalten ist.

Im Algorithmus von Hoppe et al. wird die auszuführende elementare Operation aus der Menge der bei der aktuellen Topologie zulässigen Operationen per Zufall ausgewählt. Wird durch die Ausführung der ausgewählten Operation ein Abstieg in der Zielfunktion erreicht, so wird diese Änderung akzeptiert. Im anderen Fall wird die Änderung wieder verworfen und eine neue zulässige Operation zufällig ausgewählt bis entweder ein Abstieg in der Zielfunktion erreicht werden konnte oder eine vorher festgelegte Schranke (maximale Anzahl der Versuche) überschritten wird. Bereits durch Anwendung dieser einfachen, zufallsgesteuerten Strategie können sehr gute Ergebnisse erzielt werden.

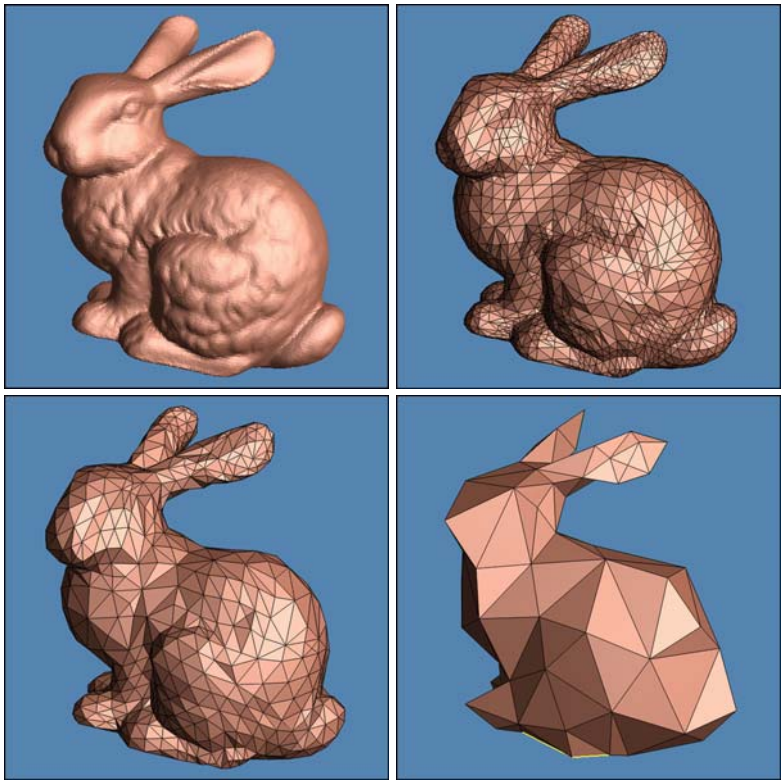

Abbildung 4-10: Verschiedene Auflösungen des Stanford Bunnys.

Abbildung 4-10 zeigt nach diesem Algorithmus erzeugte verschiedene Auflösungen des Stanford Bunnys (links oben: 70.000 Dreiecke, rechts oben: 4.600 Dreiecke, links unten: 2.000 Dreiecke, rechts unten: 162 Dreiecke).

4.2.3 Umsetzung und Ergebnisse

Die prototypische Umsetzung der agentenbasierten Steuerung der Visualisierungspipeline wurde komplett komponentenorientiert in Java entwickelt. Es kamen dabei zu Evaluierungs- und Demonstrationszwecken medizinische Volumendaten zum Einsatz, die Anwendung ist jedoch nicht auf solche Daten beschränkt. Im speziellen wurde hier der im VTK-Paket enthaltene CT-Datensatz eines menschlichen Kopfes verwendet. Der Datensatz besteht aus 94 Schichtaufnahmen mit einem Abstand von ca. 1.5 mm. Die einzelnen Graustufenaufnahmen haben eine Auflösung von 256^2 Pixel mit einem Abstand von jeweils ca. 0.8 mm und einer Farbtiefe von 12 Bit. Abbildung 4-11 zeigt eine Schichtaufnahme dieses Datensatzes.

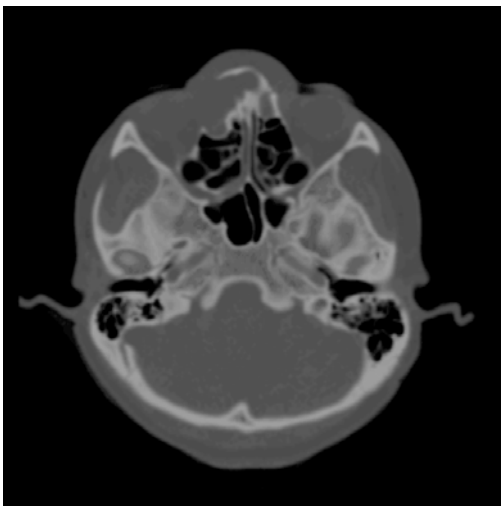

Abbildung 4-11: CT-Schichtaufnahme durch einen menschlichen Kopf.

Zunächst werden bei dieser Anwendung Dreiecksnetze aus Iso-Surfaces mittels eines Marching Cubes-Algorithmus erzeugt bzw. in einem Preprocessing-Schritt vorberechnet. Analog zu den Überlegungen aus Abschnitt 4.1.1 werden verschiedene LOD-Auflösungen (bezogen auf Geometrie und Textur) entsprechend den anwenderspezifischen Vorgaben und den tatsächlichen, gerade aktuellen Systemleistungsfähigkeiten angezeigt. Je nach Interaktion des Benutzers mit dem System und der eingesetzten Hardwarekonfiguration variiert die Visualisierungsqualität automatisch von einer sehr niedrigen Auflösung mit abgeschaltetem Texturing und abgeschalteter Beleuchtung bis hin zu höchstmöglich aufgelösten Geometrien und Texturen.

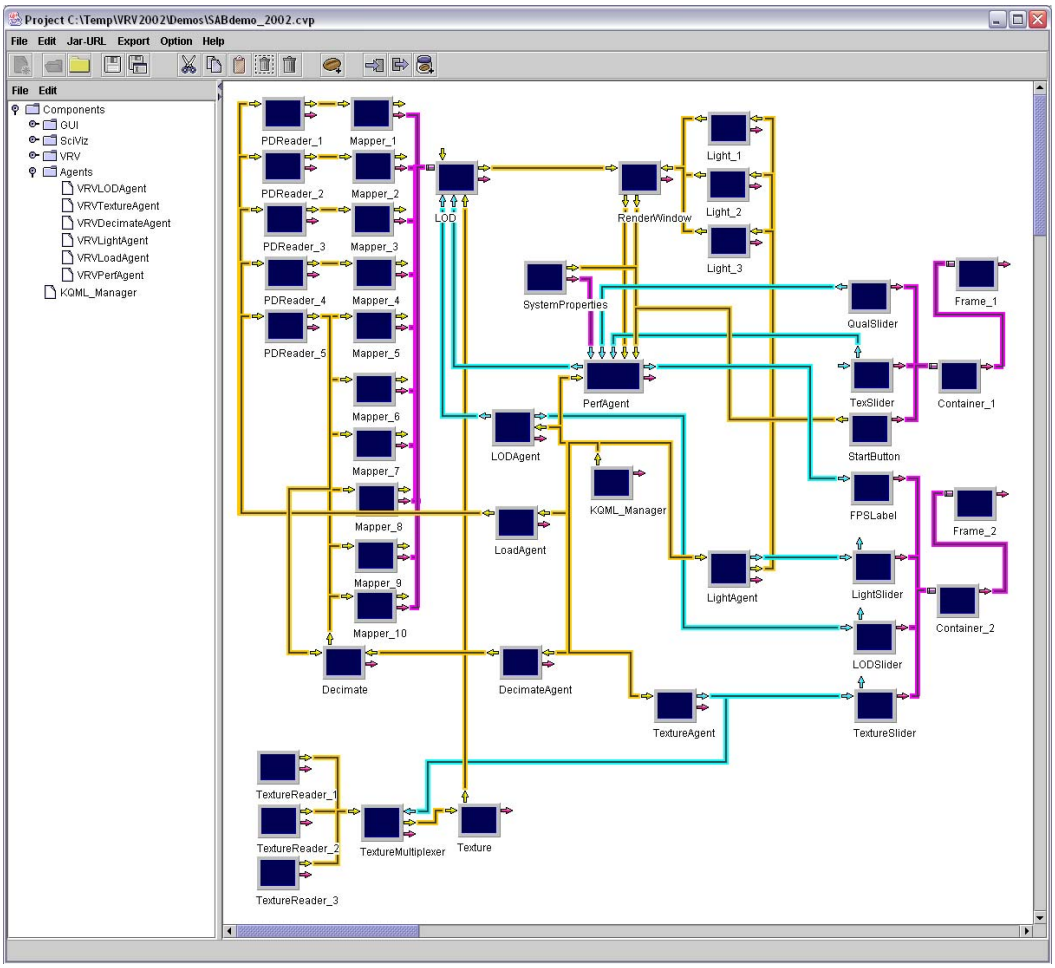

Abbildung 4-12: Visual Prototyping einer agentengesteuerten Anwendung.

Abbildung 4-12 zeigt einen Snapshot der Anwendung im Visual Prototyping System CVP. Hiermit lässt sich gut der interne Aufbau der Applikation, die Steuerung durch das Agentensystem sowie der zugehörige Daten- und Kontrollfluss nachvollziehen.

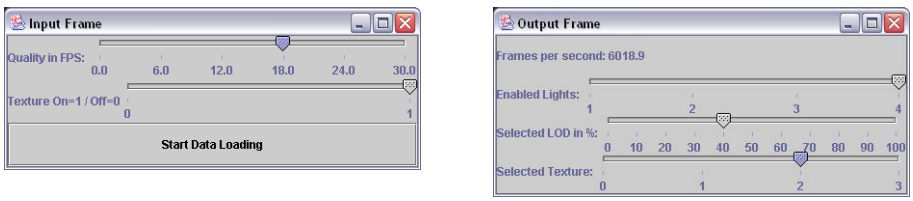

Abbildung 4-13: Benutzereingabe- bzw. Kontrollfenster.

Die Beans, welche in den ersten beiden Spalten von rechts angeordnet sind, dienen zur Erzeugung der beiden Benutzereingabe- bzw. Kontrollfenster (siehe

Abbildung 4-13), welche zur Vorgabe des Verhältnisses Qualität zu Framerate sowie zur Verdeutlichung der gerade vom Agentensystem gewählten Einstellungen dienen.

Im oberen linken Bereich des Visual Prototyping Bereiches sind fünf Beans angeordnet, die vorberechnete Auflösungsstufen einlesen (PDReader). Ausgehend von der durch PDReader_5 geladenen Geometrie werden durch die Decimate-Bean dann weitere fünf, speziell an das aktuell vorhandene System angepasste Auflösungsstufen berechnet. Die insgesamt zehn Geometrien werden an Mapper-Objekte gereicht und liefern die Eingabe für die nachgeschaltete LOD-Bean, die analog eines Multiplexers eine Geometrie mit Textur an das RenderWindow zur Visualisierung übermittelt. Die Textur wird in der beschriebenen Umsetzung aus drei möglichen, verschieden aufgelösten Texturen nach einem analogen Mechanismus ausgewählt (Bereich links unten). Zusätzlich zu Geometrie und Textur sind an die RenderWindow-Komponente noch drei (abschaltbare) Lichtquellen zur Beleuchtung der Szene angekoppelt. Betrachtet man nur die bisher beschriebenen Komponenten zusammen mit den entsprechenden Daten- bzw. Steuerleitungen, so ist hierdurch die Visualisierungsapplikation und damit auch die zugehörige Pipeline bereits komplett aufgebaut. Der Benutzer könnte nun manuell über die automatisch erzeugten Customizing-Dialoge der einzelnen Beans die Parameter für einen bestimmten Visualisierungszustand setzen. Möchte er beispielsweise den Datensatz in bester Qualität (73.567 Eckpunkte, siehe Abbildung 4-14) betrachten, so muss er bei der LOD-Bean den entsprechenden Geometrie-Level, beim TextureMultiplexer die am höchsten aufgelöste Textur und bei den Lichtern die Aktiv-Stellung auswählen.

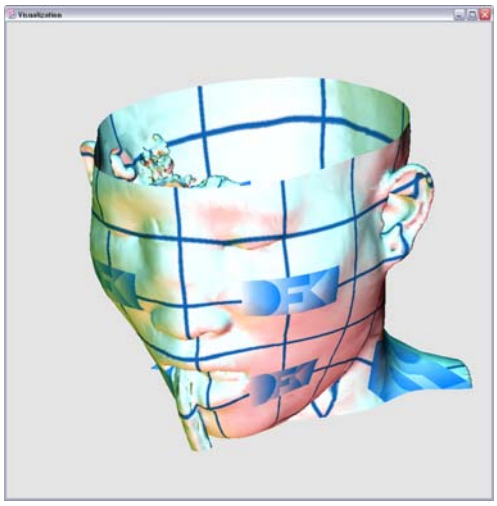

Abbildung 4-14: Visualisierung eines CT-Datensatzes (beste Qualität).

Will der Benutzer nun aber mit dem dargestellten Objekt interagieren und benötigt daher eine (zumindest einigermaßen) flüssigen Bewegungsablauf, so muss er genau diese Parameter von Hand auf für die Interaktion besser geeignete Werte setzen, welche natürlich von System zu System mehr oder weniger stark abweichen und sich dynamisch durch sich ändernde Systemlasten zudem ändern können. Hat der Benutzer dann an die gewünschte Stelle navigiert, so muss er wiederum manuell auf die höhere Qualitätsstufen zurückschalten. Es wird hier schnell klar, dass dies bereits bei einer einfachen Applikation und nur einer betrachteten Plattform ein sehr zeitaufwendiges und unpräzises Verfahren darstellt.

Die dynamische Kontrolle und Steuerung der Visualisierung soll nun aber automatisch durch das integrierte Multi-Agenten-System erfolgen, welches in Abbildung 4-12 bereits enthalten und mit den Visualisierungskomponenten verknüpft ist. Es soll an dieser Stelle bemerkt werden, dass die Agenten ebenfalls nach den Java Beans Design Pattern umgesetzt wurden und sich somit nahtlos in das Gesamtsystem einbetten lassen. Die zentrale Komponente stellt dabei der Performance-Agent dar, welcher als Eingaben die Benutzereinstellungen, die aktuell vorhandenen Systemparameter (SystemProperties-Bean) und die aktuelle Framerate vom RenderWindow erhält. Der Performance-Agent steuert basierend auf diesen Eingaben dann die einzelnen reaktiven Agenten (LoadAgent, DecimateAgent, LODAgent und LightAgent) über KQML-Kommandos an. Diese justieren entsprechend den erhaltenen Vorgaben die Parameter der jeweils zugeordneten Visualisierungsbausteine.

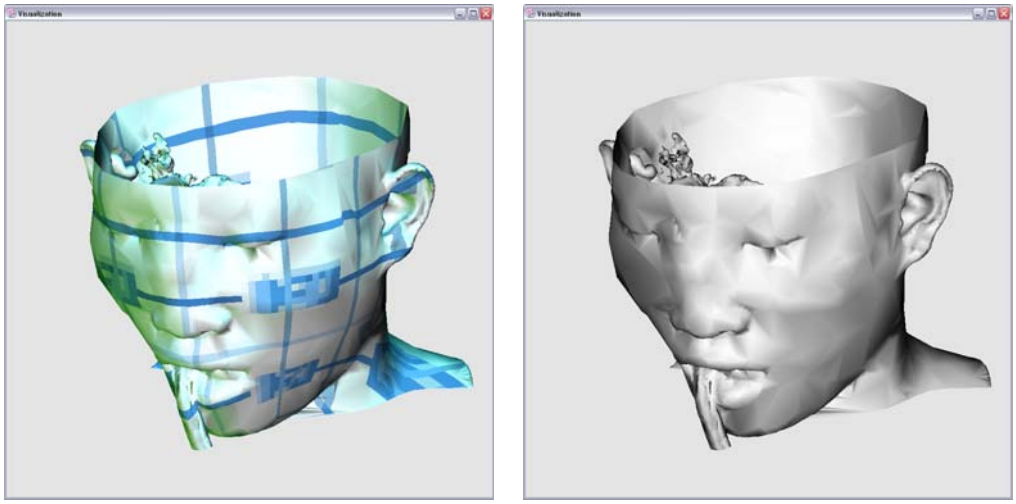

Abbildung 4-15: Visualisierung eines CT-Datensatzes (verschiedene LODs).

Abbildung 4-15 zeigt zwei Snapshots des Kopf-Datensatzes, im linken Bild mit einer Auflösung von 29426 Punkten (ca. 40% der höchsten Auflösung), im rechten Bild mit einer Auflösung von 17449 Punkten (ca. 20%). Während bei der linken

Darstellung die Texturen, wenn auch in einer niedrigen Auflösung, noch aktiviert sind, sind diese rechts aus Performancegründen vom Agenten ausgeblendet worden.

4.3 Mobile Visualisierung

Bei der in den vorangegangenen Abschnitten beschriebenen Technik der agentenbasierten Steuerung der Visualisierungspipeline zur Erzeugung einer adäquaten, interaktiven Darstellung von komplexen Objekten auf unterschiedlich leistungsfähigen Plattformen bzw. verschieden stark ausgelasteten Systemen, stellt sich direkt die Frage nach der Relevantheit solcher Methoden in der Zukunft. Ein Gegenargument für den Einsatz der beschriebenen Verfahren stellt sicher die Aussage dar, dass die Rechner immer leistungsfähiger werden und daher die interaktive Visualisierung von Datensätzen wie dem oben beschriebenen CT-Datensatz auf künftigen Rechnern kein Problem mehr darstellen wird. Dem lassen sich jedoch zwei gewichtige Argumente entgegen halten.

Zum einen steigen mit neuen Rechnergenerationen immer auch die Ansprüche der Benutzer. So ist zu beobachten, dass sich im Vergleich zur Entwicklung der Leistungsfähigkeit der Rechner zeitgleich die darzustellenden Datenmengen sogar überproportional vergrößert haben und auch deren Komplexität weiter rapide zunimmt. Es wird damit klar, dass die im Rahmen dieser Arbeit eingeführten Techniken und Methoden zur dynamischen Kontrolle und Steuerung der Visualisierungspipeline auch in Zukunft ihre Berechtigung haben werden.

Neben den stationären PCs werden mobile Geräte wie Laptops oder in jüngerer Zeit verstärkt auch PDAs und Handys für (Visualisierungs-)Anwendungen immer interessanter. Zentrales Argument ist hier die Verfügbarkeit der eigenen Daten und Informationen zu jeder Zeit und an jedem Ort. Die Leistungsfähigkeit mobiler Geräte, vor allem die der PDAs und der Handys, hinkt jedoch der der stationären Geräte deutlich hinterher. Auf diesen Umstand adaptierende flexible Anwendungen erzwingen daher geradezu den Einsatz der beschriebenen Techniken zur dynamischen Kontrolle der Visualisierung. Neben den Performanceaspekten ist aber auch die Art des Ausgabemediums, d. h. der Darstellungskontext, ein entscheidender Faktor. So muss je nach Art, Größe und Auflösung des aktuell verwendeten Gerätes die darzustellende Information verschieden aufbereitet werden. Es wird in diesem Zusammenhang zudem deutlich, dass auch Client-Server-Ansätze in das Konzept aufgenommen werden müssen, da sich die verfügbaren mobilen Geräte sowohl von ihrer Rechen- als auch von ihrer Speicherleistung ansonsten nicht für anspruchsvolle Visualisierungsaufgaben eignen würden.

Die folgenden Betrachtungen beschäftigen sich mit der skalierbaren Informationsvisualisierung auf mobilen Endgeräten, dargestellt am Demonstrator der effizienten Informationsvermittlung und betrieblichen Leistungssteigerung im Bereich des Betriebs von Abwasseranlagen. Der Fokus der Arbeiten lag dabei auf der Konzeption und Entwicklung eines Frameworks sowie eines Technologiedemonstrators für ein flexibles, mobiles Informationsvisualisierungssystem. Anhand des Demonstrators wurden beispielhaft mögliche Anwendungen im Rahmen der Betriebsführung praktisch umsetzt.

Der Einsatz eines mobilen Visualisierungssystems ist in den verschiedensten Bereichen denkbar, die sich nicht immer vollständig voneinander trennen lassen. Geeignete Einsatzmöglichkeiten finden sich unter anderem in den Bereichen:

- Prozessüberwachung, -visualisierung und -optimierung,
- Controlling und Benchmarking,
- Reparatur und Wartung,
- innerbetriebliche Aus-, Fort- und Weiterbildung sowie
- Arbeitssicherheit.

Im Folgenden werden zunächst die technologische Ausgangslage und die Anwendungsmöglichkeiten eines mobilen Informationssystems bei der Betriebsführung von Abwasseranlagen dargestellt und die Konzeption und Umsetzungsstrategie für ein solches System beschrieben. Weiterhin wird die technische Umsetzung einiger beispielhaften Anwendungen in Form eines Technologiedemonstrators vorgestellt.

Die hier aufgeführten Forschungsarbeiten sind Bestandteil mehrerer wissenschaftlicher Veröffentlichungen ([101],[110]) auf nationalen und internationalen Konferenzen. Sie sind Teil des von der Stiftung Rheinland-Pfalz für Innovation geförderten Forschungsprojekts *Similar.*

4.3.1 Technologische Ausgangslage

Die Siedlungswasserwirtschaft hat im Bereich der Abwasserableitung und Abwasserbehandlung in den letzten Jahren eine enorme Entwicklung vollzogen, deren Ende zurzeit noch nicht absehbar ist. Steigende Anforderungen an die Qualität der Abwassereinleitungen in Kombination mit im Einzelfall sehr unterschiedlichen Randbedingungen haben dazu geführt, dass eine Vielzahl unterschiedlicher Verfahrenstechniken zur Ableitung und Behandlung von Abwasser eingesetzt werden. Hinzu kommt, dass ein Abwassernetz aus vielen räumlich verteilten Teilsystemen besteht, die zwar zunächst unabhängig voneinander dezentral lokale Aufga-

ben durchführen, untereinander aber verfahrenstechnisch eng verknüpft sind und so innerhalb eines Gesamtsystems aufeinander einwirken.

Entsprechend vielschichtig und kompliziert ist auch die Betriebsführung von Abwasserbeseitigungssystemen geworden, da das Personal sich sowohl mit zahlreichen höchst dynamischen physikalischen, chemischen und biologischen Vorgängen auseinander setzen muss, als auch über Kenntnisse in den Bereichen der Maschinen-, Mess- und Elektrotechnik sowie in rechtlichen und zunehmend auch in wirtschaftlichen Fragen verfügen muss. Des Weiteren sind in der täglichen Arbeit zahlreiche Aspekte der Arbeitssicherheit zu berücksichtigen.

Diese Situation führt häufig zu einem suboptimalen Betrieb von Abwasseranlagen sowie zu suboptimalen Arbeitsabläufen und bringt damit Nachteile im Bereich der Betriebskosten und für den Gewässerschutz mit sich.

Zwar unterstützen heute in vielen Fällen Prozessleitsysteme (PLS) die Betriebsführung von Abwasseranlagen, indem z. B. Aufgaben des Steuerns und Regelns automatisiert und Prozessdaten dokumentiert und dargestellt werden. Dennoch ist das Betriebspersonal in der täglichen Arbeit auf viele verschiedene Informationsquellen angewiesen, wie z. B. auf (oft nur in Papierform verfügbare) Literatur und Bedienungsanleitungen sowie auf das (meist mündlich überlieferte) Know-how von Kollegen und gegebenenfalls von Experten. Hinzu kommt, dass viele Informationen nicht fortlaufend aktualisiert werden und nur schwer erkennbar ist, ob sie den aktuellen Stand widerspiegeln. Weiterhin werden viele Arbeiten auf Abwasseranlagen außerhalb von Büroräumen oder der Schaltwarte verrichtet, also entfernt von den Räumen, in denen Informationen verfügbar sind und gewartet werden.

Parallel zu den immer komplexer werdenden Anforderungen an das Betriebspersonal von Kläranlagen führt seit einigen Jahren eine rasante Entwicklung bei mobilen Visualisierungsgeräten dazu, dass diese in Form von Laptops, Pocket PCs oder Mobiltelefonen zunehmend erschwinglicher und verbreiteter werden. Es liegt daher nahe, diese Geräte zur mobilen Visualisierung von Informationen zur Betriebsführung von Abwasseranlagen einzusetzen.

In der Fachliteratur ist nur wenig über Möglichkeiten der bedarfsgerechten mobilen Darstellung von verfügbaren und aufbereiteten Daten zu finden. Einen Ansatz hierzu liefern Schütze et al. [102]. Die Verfasser haben zwei Prototypen entwickelt, die Messdateninformationen von Kläranlagen auf mobile Endgeräte transferieren. Mittels Telekommunikation über WAP (Wireless Application Protocol) und mit Hilfe der Beschreibungssprache WML (Wireless Markup Language) wurden auf Mobiltelefonen die wichtigsten Prozessinformationen verfügbar gemacht. Der zweite Prototyp stellte eine ähnliche Funktionalität mittels der Beschreibungssprache HTML (Hypertext Markup Language) auf Pocket PCs zur Verfügung. Das

Gesamtkonzept beschränkt sich aber hauptsächlich auf die Übertragung von Messdaten. Weitergehende Aufgaben und Informationen wie Wartungs- / Reparaturanleitungen, Sicherheitshinweise oder allgemeine Informationen zu Aggregaten und Bauwerken sind nicht enthalten. Die Entwickler konnten mit ihren Untersuchungen aber zeigen, dass mobile Online-Informationssysteme zu einem effizienteren Betrieb von Abwasseranlagen beitragen können.

Die im Bereich der Informationsvisualisierung eingesetzten Systeme stellen i. A. weit reichende Anforderungen an die zu verwendende Hardware. Ein Einsatz auf mobilen Geräten ist aufgrund der bislang stark eingeschränkten Performance und grafischen Darstellungsmöglichkeiten zwar möglich, es fehlen jedoch geeignete Optimierungen des Visualisierungsprozesses für komplexe Darstellungen. Um eine Vielzahl mobiler Geräte mit unterschiedlichen Displaygrößen und Rechenleistungen zur Informationsvisualisierung einsetzen zu können, müssen flexible Client-Server-Architekturen benutzt werden. Der Server muss hierbei in der Lage sein, die Rechenlast und die Menge der zu übertragenden Daten flexibel an den jeweiligen Leistungsumfang des mobilen Gerätes anzupassen. Im Folgenden wird ein kurzer Überblick der relevanten mobilen Endgeräte gegeben.

Zu den kleinsten Geräten, die prinzipiell in Frage kommen, zählen derzeit Mobiltelefone mit eigenem Betriebssystem und einer Java Virtual Machine. Einige dieser Geräte verfügen sogar über die Möglichkeit der Bild- oder Videoaufzeichnung, ein grafisches Farbdisplay gehört schon fast zum Standard. Die Rechenleistung und die Speicherkapazität ist aber noch auf ein Minimum begrenzt.

Eine höhere Leistungsklasse bilden die Pocket PCs, auch PDAs (Personal Digital Assistant) genannt. Mit großen Displays und Touchscreen-Fähigkeit bei insgesamt kompakten Abmessungen erfüllen auch sie die prinzipiellen Anforderungen der mobilen Informationsvisualisierung. Die Rechenleistung ist ausreichend für einfache Office-Anwendungen, reicht jedoch nur teilweise für komplexe Informationsvisualisierung aus. In naher Zukunft sind Geräte zu erwarten, die wie Desktop-Computer über integrierte 3D-Hardwarebeschleunigung verfügen und damit auch die Darstellung dreidimensionaler Konstruktionszeichnungen o. ä. in Echtzeit ermöglichen.

Eine interessante neue Klasse mobiler Geräte bilden die Tablet PCs. Diese können mit Hilfe eines Stiftes bedient werden. Die Eingabe von Text erfolgt mit Hilfe einer recht zuverlässigen Handschrifterkennung. Für die mobile Visualisierung sind Tablet PCs aufgrund ihres handlichen Formates, ihres neuen Eingabekonzeptes und einer akzeptablen Rechenleistung interessant.

Voraussetzung zur mobilen Visualisierung ist natürlich eine drahtlose Datenübertragung. Im Folgenden sind die wichtigsten Techniken kurz dargestellt.

Aktueller Stand der Technik bei drahtlosen Fernverbindungen sind GPRS (General Packet Radio Service) und das noch im Aufbau befindliche UMTS (Universal Mobile Telecommunication Standard). Erste UMTS-fähige Handys stehen kurz vor der Auslieferung, beispielsweise das Nokia 6650 und das Siemens U15. Die wichtigsten Netzbetreiber geben als Starttermin für UMTS-Dienste in Deutschland Anfang bis Mitte 2004 an.

Für lokale drahtlose Nahverbindungen sind sowohl Wireless Local Area Network (WLAN) als auch Bluetooth geeignet. Beide Standards erfahren zurzeit eine starke Verbreitung und zunehmende Integration in das gesamte Spektrum mobiler Geräte. WLAN stellt das drahtlose Äquivalent zum weit verbreiteten Ethernet dar und besitzt aus Sicht des Anwenders ähnliche Eigenschaften. Die Reichweite kann bis zu ca. 200 m betragen, was den Einsatz auf Kläranlagen möglich macht. Zum Betrieb ist nur die Aufstellung eines zentralen Access Points notwendig, der an einen Server angeschlossen ist. Alle WLAN-fähigen mobilen Geräte können dann im Empfangsbereich mit Daten versorgt werden, die der Server bereitstellt.

Im Gegensatz zu WLAN ist Bluetooth als universeller Ersatz für Drahtverbindungen gedacht; so definiert der internationale Bluetooth-Standard zahlreiche verschiedene „Profiles“ für das jeweilige Einsatzgebiet. Die Reichweite und auch die Übertragungsrate von Bluetooth sind zwar im Gegensatz zu WLAN geringer (bis ca. 20 m), die Funktionsvielfalt eröffnet aber neue Anwendungsfelder. Mit dem „Service Discovery Profile“ etwa kann ein Bluetooth-fähiges Gerät erkennen, welche Dienste ein anderes Bluetooth-Gerät bereitstellt. Weitere Profile erlauben die Übertragung von Bildern und Sprache. Aufgrund der sinkenden Kosten von Bluetooth-Chips und des niedrigen Energieverbrauchs dieser Technik ist weiterhin eine starke Verbreitung zu erwarten, die sich aller Voraussicht nach auch auf den Bereich von Messinstrumenten und anderen Elektroinstallationen erstrecken wird. Auch hier eröffnen sich damit Anwendungsfelder, die für Wartungsaufgaben mit mobilen Geräten eine Rolle spielen werden.

Die Integration von mobilen Positionierungssystemen wie etwa GPS ist bisher nur vereinzelt bei mobilen Anwendungen anzutreffen. Eine solche Integration ermöglicht z. B. ein „Situational Visualization System“, das abhängig vom aktuellen Aufenthaltsort des Anwenders die Informationen entsprechend aufbereitet. Erste mobile Pocket PCs mit eingebautem GPS-Empfänger sind bereits verfügbar, z. B. der iQue 3600 von Garmin. Das Zusammenspiel von mobilen Kleinstcomputern und Positionierungssystemen ist Gegenstand von nur wenigen Untersuchungen. Hier besteht noch geräumiger Entwicklungsspielraum bei der Konzeption mobiler Anwendungen.

4.3.2 Anwendungsmöglichkeiten mobiler Visualisierung

Die folgenden Punkte beschreiben Anwendungsmöglichkeiten für ein mobiles Visualisierungssystem im Kontext von Abwasseranlagen.

4.3.2.1 Arbeitssicherheit

Für viele Arbeiten, die auf Abwasseranlagen durchgeführt werden, sind Arbeitssicherheitsvorschriften zu beachten, die meist zentral in der Leitwarte einer Kläranlage in gedruckter Form vorgehalten werden. Mit Hilfe eines mobilen Informationssystems können den Mitarbeitern Hinweise (z. B. 3D-Darstellung von Befestigungspunkten für Absturzsicherungen), Vorschriften, Anweisungen und Hilfen (z. B. Telefonschnellwahl im Notfall) online an jedem beliebigen Ort zur Verfügung gestellt werden. Dies ist gerade für Mitarbeiter relevant, bei denen bestimmte Arbeitsschritte noch nicht routiniert ablaufen.

Die Unfälle der Vergangenheit haben ferner gezeigt, dass Vorgesetzte bei Unfällen in Beweisnot geraten, wenn sie nicht nachweisen können, dass die Vorschriften beachtet bzw. dass Mitarbeiter über mögliche Gefahren ausreichend informiert wurden. Zur Information des Mitarbeiters und zur Beweissicherung kann das System eine Checkliste mit Quittiersystem enthalten, die vor einer gefahrvollen Tätigkeit von den Mitarbeitern abgearbeitet werden muss.

4.3.2.2 Reparatur, Wartung und Störungen

Bei den heutigen abwassertechnischen Anlagen werden zahlreiche Maschinen und Aggregate eingesetzt, die einer regelmäßigen Wartung und Pflege bedürfen, was zu einem hohen Reparatur- und Wartungsaufwand führt. Bei Arbeiten im Bereich von Abwasseranlagen ist es außerdem oft nicht oder nur in eingeschränktem Maße möglich, am Arbeitsort auf reparatur- und wartungsrelevante Daten zurückzugreifen. Ein mobiles Informationssystem ermöglicht es dem Personal, erforderliche Daten und Arbeitsabläufe vor Ort abzufragen und auch von unterwegs auf zentrale Daten oder das PLS zurückzugreifen.

Auch ist es denkbar, dass das Personal bei einem Inspektionsgang in optischer und / oder akustischer Form darüber informiert wird, dass ein Gerät, an dem der Mitarbeiter gerade vorbei geht, gewartet werden muss.

Bei nicht permanent besetzten Anlagen kann ein mobiles System der Rufbereitschaft, z. B. bei Störungen, weitere Hintergrundinformationen liefern. Die Mitarbeiter können, ohne vor Ort zu sein, entscheiden, ob ein sofortiges Eingreifen erforderlich ist oder ob die Störung gegebenenfalls vom mobilen Endgerät aus behoben werden kann. Eine Lösung auf Mobilfunkbasis erscheint somit sinnvoll und kann zu einer Reduzierung des erforderlichen Aufwandes beitragen.

4.3.2.3 Prozessüberwachung, -visualisierung und -optimierung

Moderne Prozessleitsysteme unterstützen die Betriebsführung von Abwasseranlagen insbesondere in den Bereichen Datenerfassung und Speicherung sowie bei Aufgaben des Steuerns und Regelns. Diese Daten sind in der Schaltwarte verfügbar. Um jedoch Beobachtungen, die z. B. während der üblichen Arbeiten auf der Kläranlage gemacht werden, vor Ort durch weitere Informationen zu ergänzen, ist eine Verfügbarkeit dieser Dateien auf mobilen Geräten erforderlich.

Das mobile Informationssystem muss daher die Informationen so aufbereiten, dass dem Benutzer die für das jeweilige Bauteil wichtigen Kenngrößen, Betriebszustände etc. optisch und / oder akustisch in einfach zugänglicher Art und Weise zur Verfügung gestellt werden. Ist das mobile System zusätzlich mit einer Digitalkamera ausgestattet, können außerdem auch Betriebszustände mittels Kurzvideo oder Foto optisch erfasst werden. Die optische Dokumentation solcher Ereignisse ermöglicht es dem Spezialisten (in Kombination mit den ebenfalls erfassten Betriebsdaten) auch im Nachhinein oder aus der Ferne, ein Problem zu untersuchen (z. B. Bläh- und Schwimmschlammprobleme).

Bei der Ermittlung des Optimierungspotenzials aus Daten und Expertenwissen kann der Einsatz von Virtuellen und Erweiterten Realitäten einen wichtigen Beitrag leisten. Aufgrund der Dezentralität vieler Anlagenteile eines Abwassersystems ist in vielen Anwendungsfällen auch ein mobiles System erforderlich, welches, ausgestattet mit einem Positionierungssystem, auch bauwerksbezogene Informationen anzeigen kann.

4.3.2.4 Benchmarking und Controlling

Ein weiterer Einsatzschwerpunkt für das mobile System liegt im technischen und betriebswirtschaftlichen Controlling. So wäre es mit Hilfe von GPS sowie einem Navigationssystem möglich, die Fahrtwege und Fahrtdauern zu Außenanlagen, Sonderbauwerken o. ä. nachzuvollziehen. Darüber hinaus kann durch den Einsatz von Check- oder To-Do-Listen in Kombination mit Quittierungssystemen die Dauer einzelner Arbeitsschritte ermitteln werden sowie Verbrauchs- bzw. Betriebsdaten manuell in das mobile System eingegeben werden. Das System meldet dann die gesammelten Daten an ein stationäres System weiter, wo sie monetär bewertet werden. Werden technische und wirtschaftliche Informationen mehrerer vergleichbarer Anlagen gesammelt und ausgewertet, können an das mobile System auch entsprechende Vergleichswerte (Benchmarking) zurückgemeldet werden. Weicht ein Ist-Wert vom Soll-Wert signifikant ab, kann das Betriebspersonal die möglichen Ursachen ermitteln.

Durch die Unterstützung des mobilen Systems können nicht nur anlagenspezifische, sondern auch (arbeits-)prozess- oder bauteilspezifische monetäre und nicht-

monetäre Kenngrößen ermittelt werden. Insbesondere die Ermittlung von prozess- und bauteilspezifischen Kosten, die in vielen Bereichen der Wirtschaft bereits erfasst werden, erfolgt in der Siedlungswasserwirtschaft kaum, da der Aufwand hierfür vielfach noch als zu hoch angesehen wird. Es liegt auf der Hand, dass die Ermittlung solcher Daten das interne Rechnungswesen sowie die Planung und Optimierung von Arbeitsprozessen in Abwasserreinigungsbetrieben einen erheblichen Schritt voran bringen würde.

4.3.2.5 Fort- und Weiterbildung

Die Vielseitigkeit und Komplexität von modernen Abwasseranlagen erfordert vom Betriebspersonal eine ständige Fort- und Weiterbildung. Häufig treten Fragestellungen gerade in Situationen auf, die eine Umstellung oder Modifizierung des Betriebes erfordern. Dann ist eine zeitnahe, problemorientierte Fortbildung erforderlich. Für den Einsatz des mobilen Informationssystems in der innerbetrieblichen Aus-, Fort- und Weiterbildung ergeben sich somit einige Möglichkeiten: Durch die Integration von Online-Hilfen, die hierarchisch gegliedert sein können, – kombiniert mit den Möglichkeiten schneller Datenübertragung – müssen sich diese Hilfen nicht nur auf das geschriebene Wort beschränken, sondern können auch auf Lernvideos und dreidimensionale Objekte mit weit reichenden Interaktionsmöglichkeiten (z. B. bei schwer zugänglichen Bauwerken) zurückgreifen.

4.3.3 Umsetzung und Ergebnisse

Das im Rahmen des Projekts erstellte System besteht aus zwei Hauptkomponenten: einer relationalen (MySQL-) Datenbank, die alle darzustellenden Informationen (außer Mess- und Betriebsdaten) enthält und einem (JSP-basierten) Web-Container. JSP (Java Server Pages) ermöglicht die dynamische Generierung von HTML-Seiten; der JSP-Code wird mit Hilfe des Tomcat-Servers ausgeführt. Durch die dynamische Seitengenerierung sind sowohl Reaktionen auf Benutzereingaben als auch die Anpassung an verschiedene mobile Endgeräte möglich. Das System-Framework des Technologie-Demonstrators ist in folgendem Überblicksbild dargestellt. Aus Gründen der Übersichtlichkeit wurde auf die Darstellung der Kommunikationsverbindungen der integrierten Agenten verzichtet.

Um auch Mess- und Betriebsdaten einer Kläranlage darstellen und auswerten zu können, ist als dritte Hauptkomponente die im PLS einer Anlage integrierte Datenbank notwendig. Die geplante Einbindung in das Framework ist ebenfalls in Abbildung 4-16 zu erkennen.

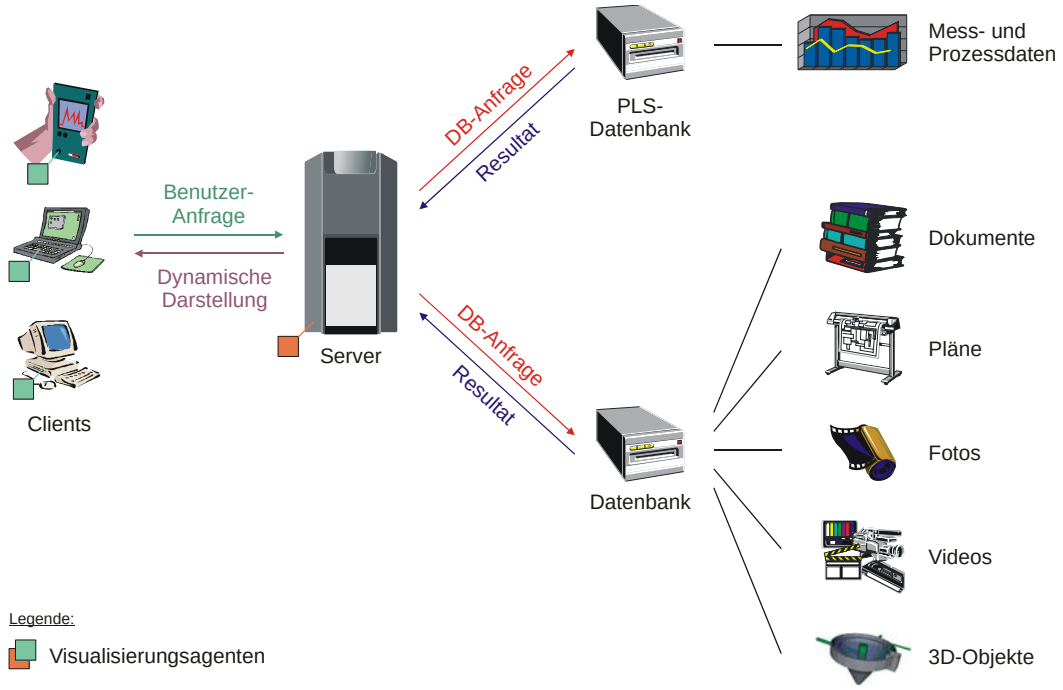

Abbildung 4-16: System-Framework des Technologie-Demonstrators.

Bei der Nutzung des Systems durch den Anwender wird die Benutzeroberfläche in einem HTML-fähigen Browser dargestellt. Bevor die Startseite angezeigt wird, werden von einem Visualisierungsagenten (reaktiver Agent) die grafischen Leistungsmerkmale des Clients festgestellt und an den anfragenden Visualisierungsagenten des Servers übermittelt. Der Server kann dann die Benutzeranfragen (Anklicken von Menüpunkten, Eingabe von Suchbegriffen) entgegennehmen und entsprechende SQL-Anfragen an die Datenbank weiterleiten. Die Ergebnisse der Datenbankabfrage werden zurück an den Server übermittelt, der für die Anzeige auf dem Endgerät eine HTML-Seite erstellt. Die in den HTML-Seiten enthaltenen Menüicons, Tabellen und sämtliche Fotos und Zeichnungen werden von dem Visualisierungsagenten des Servers automatisch an die vorhandene Displaygröße und Leistungsfähigkeit des Clients angepasst. Insbesondere werden Fotos vor der Übertragung skaliert, so dass nicht mehr Daten übertragen werden als tatsächlich benötigt werden. Sollen Videos dargestellt werden, müssen diese in den entsprechenden Auflösungen und Bitraten in der Datenbank abgelegt werden, damit zur Laufzeit eine optimal an die Hardware abgestimmte Version ausgewählt werden kann. Ein geeignetes Videoformat stellt MPEG dar, da dies von vielen Systemen abgespielt werden kann.

Das hier verwendete Framework ist nicht auf HTML beschränkt, es können auch andere Ausgabeformate wie z. B. WML für Mobiltelefone verwendet werden. Wie in Abbildung 4-16 angedeutet, ist die Einbindung von Mobiltelefonen als Weiter-

entwicklung geplant. Während WML mittlerweile von fast allen aktuellen Mobiltelefonen unterstützt wird, gibt es für neuere Handys auch die Möglichkeit, HTML-Browser auszuführen. Die Ausführung des Systems auf einem derart kleinen Display mit einer entsprechend geringen Auflösung ist jedoch in Hinblick auf die Praxistauglichkeit momentan eher als nicht sinnvoll zu bewerten.

Das oben beschriebene Framework wurde im Rahmen eines Technologiedemonstrators für die Kläranlage Meisenheim (Rheinland-Pfalz, 14600 Einwohner) beispielhaft in die Praxis umgesetzt. Folgende Funktionen bzw. Informationen werden dabei vom System bereitgestellt:

- Liste aller Bauwerke und der darin enthaltenen Aggregate
- Fotos und Pläne der gesamten Anlage, der meisten Bauwerke und der wichtigsten Aggregate
- eine Liste der wichtigsten Abwasser-Reinigungsprozesse, der zugehörigen Messwerte und eine Funktion zur kurzen Beurteilung der Reinigungsleistung
- Wartungsanleitungen und Anleitungen zur Behebung von Störfallen am Beispiel einer Rücklaufschlammpumpe mit ergänzenden Fotos, Zeichnungen und Videos
- Fortbildungsdokumente mit grundlegenden Informationen zur biologischen Abwasserreinigung
- Notfallfunktion mit Anweisungen und Bildern zur Ersten Hilfe
- Suchfunktion zum Durchsuchen des Informationsbestandes

Als Server des Testsystems kommt ein Rechnersystem mit Pentium IV 2.6 GHz Prozessor und 1 GB RAM zum Einsatz. Die relationale Datenbank läuft auf MySQL in der Version 4.0.14, als JSP-Server wird Tomcat in der Version 4.0.6 eingesetzt.

Das erste mobile Testgerät ist ein iPAQ H5550 mit Intel XScale PXA 255 Prozessor und Windows Mobile 2003 als Betriebssystem. Internet-Browser ist der Pocket Internet Explorer, die Anbindung an den Server geschieht über die integrierte WLAN-Funktionalität. Das Abspielen der Videos erfolgt mit Hilfe der Software PocketTV. Mit Hilfe des Touchscreens lässt sich problemlos durch das System navigieren, eine Skalierung der Tabellen ermöglicht eine komfortable Übersicht. Insbesondere bei langen Tabellen muss jedoch von den Scroll-Leisten Gebrauch gemacht werden Die Eingabe von Suchbegriffen kann problemlos über die eingeblendete Tastatur erfolgen. Das Abspielen der Videos im MPEG-1 Format geschieht bei einer Auflösung von 320x240 Pixel ruckfrei und in sehr guter Qualität.

Zweites Testgerät ist ein Toshiba Portegé P3500. Dies ist ein so genanntes Convertible, also eine Kombination aus Tablet PC und gewöhnlichem Notebook. Die Anbindung an den Server erfolgt über eine WLAN-Verbindung mit 11 MBit Bandbreite. Als Browser wird der Internet Explorer der Windows XP Tablet PC Edition verwendet. Die Bedienung ist sowohl im Notebook- als auch im Tablet PC-Modus durchführbar. Im Tablet-Modus gelingt das Anklicken von Schaltflächen zielsicherer, dafür ist die Texteingabe beispielsweise in den Suchen-Feldern etwas schwieriger. Die Geschwindigkeit der Darstellung ist optimal. Das Abspielen von Videos übernimmt der Windows Media Player.

Die folgenden Abbildungen veranschaulichen die Funktionsweise der einzelnen Menüpunkte. Es sind jeweils links die Ansicht auf einem Desktop- / Notebook-System und rechts die Ansicht auf einem Pocket PC-Systems dargestellt. Die Darstellung der Information, z. B. in Bezug auf Darstellungsgröße, Auflösung, Dateiformat, Videobitrate, Level of Detail-Einstellung bei 3D-Objekten usw., wird dabei durch den Visualisierungsagenten automatisch an den jeweiligen Client angepasst.

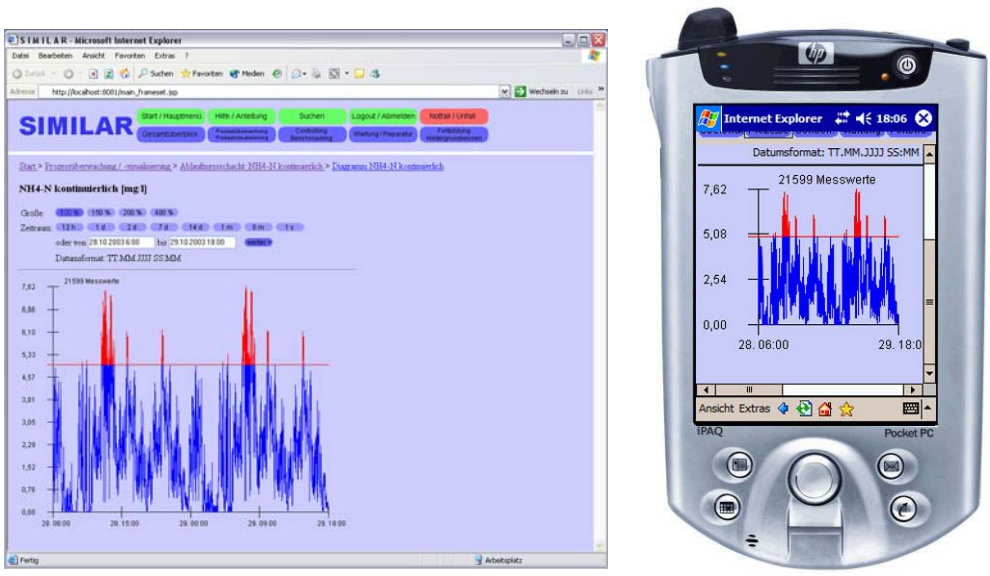

Abbildung 4-17: Prozessüberwachung und -visualisierung.

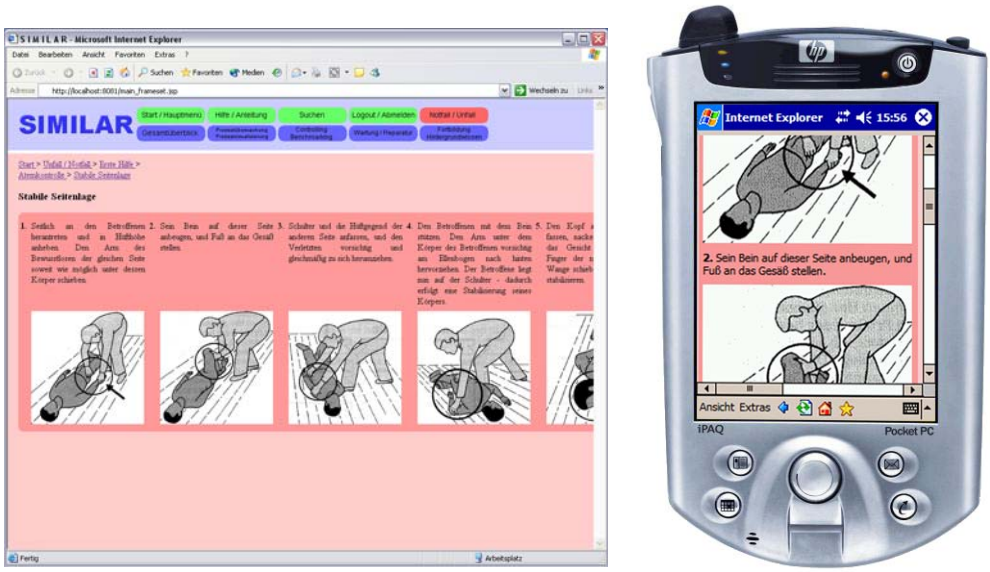

Abbildung 4-18: Unterstützung der Arbeitssicherheit (Notfallanweisungen).

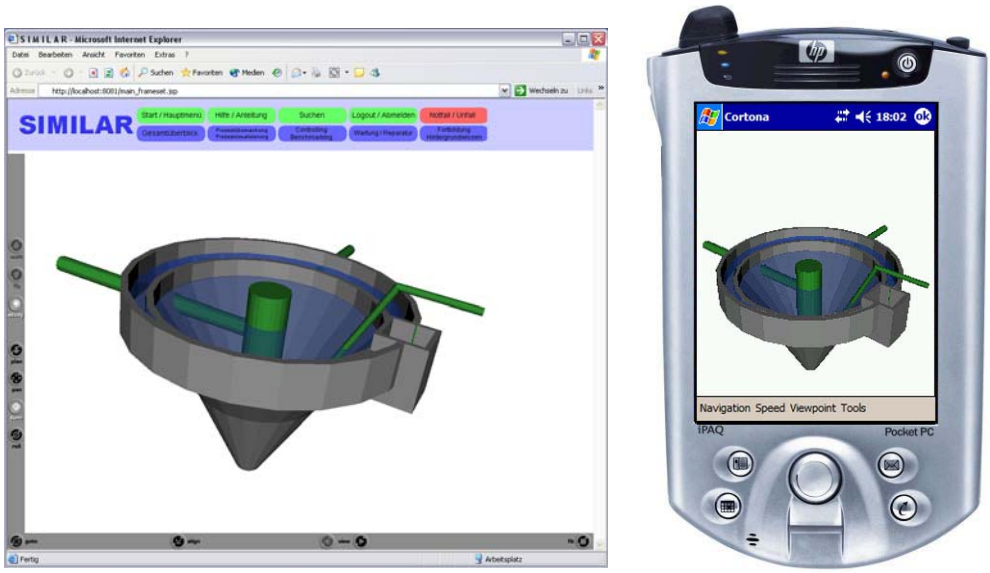

Abbildung 4-19: Visualisierung 3-dimensionaler Daten (Sandfang).

Abbildung 4-20: Präsentation von Videomaterial (Wartung einer Pumpe).

5

VIRTUAL TRY-ON – Daten- und Benutzerkontext

Bekleidungsindustrie und Bekleidungshandel sind mit über 50 Milliarden Euro Jahresumsatz in Deutschland ein wichtiger Konsumgütermarkt. Dabei zeigt der Kunde gerade in diesem Bereich zunehmend das Bedürfnis nach Individualität und Service: Einerseits hat der Kunde den Wunsch, möglichst zu jeder Zeit und an jedem Ort bequem und preiswert einkaufen zu können. Ein Beispiel hierfür ist der traditionelle Einkauf von Textilien per Katalog. Zum anderen fordert der Kunde ein breites Produktsortiment bei einer ansprechenden Produktpräsentation und bester individueller Anpassung sowie Orientierung an seinen Bedürfnissen. Zugleich fordert der Kunde aber auch möglichst niedrige Kosten, wodurch Firmen immer häufiger zum Ausweichen auf die Produktion im Ausland gezwungen werden.

Einen möglichen Ausweg aus der geschilderten Problematik bietet hier die industrielle Fertigung individualisierter Produkte für den Massenmarkt. Die grundlegende Idee dieses Konzeptes besteht darin, durch neuartige Organisationsstrukturen und Vertriebswege in Verbindung mit dem Einsatz hoch entwickelter Informationstechnik die Produktion und Vermarktung individuell auf den Kunden zugeschnittener Produkte zu Preisen im Bereich herkömmlicher Massenartikel zu ermöglichen. Für den Handel bietet die Maßproduktion ein geringeres Lagerrisiko, Liquiditätsgewinn und eine stärkere Kundenbindung. Die Vorteile für den Kunden sind eine gesteigerte Produktqualität und -identifikation. Der Nutzen für

den Hersteller besteht in der Reduktion des Produktrisikos und der schnellen Anpassung an aktuelle Kundenanforderungen.

Voraussetzung für die beschriebene Produktindividualisierung ist zunächst die Bereitstellung grundlegend neuer Technologien für die Produktpräsentation, -auswahl, -anpassung und -anprobe. Im Zentrum steht dabei die Virtualisierung klassischer Abläufe durch intensiven Einsatz von Methoden der Virtual Reality. Ziel ist die Schaffung von virtuellen Verkaufsräumen (sowohl im Bekleidungsgeschäft als auch zu Hause), die das Anbieten sowohl von Produkten mit normalen Konfektionsgrößen als auch individualisierter Produkte unterstützen.

Eine wichtige Voraussetzung bei der Erschließung alternativer Vertriebswege stellt heute vor allem die Entwicklung des Internets von der reinen Informationsplattform zu einem Medium dar, über das Produkte und Dienstleistungen aller Art vertrieben werden können. Diese Form des Handels wird unter dem Schlagwort Electronic Commerce (E-Commerce) zusammengefasst.

Das zentrale Ziel der beschriebenen Virtual Try-On Anwendung liegt daher in der Konzeption und Umsetzung eines interaktiven Individual-Katalogs, d. h. einem Online-Katalog, in dem der Kunde sich selbst als 3D-Modell in der gewählten Bekleidung betrachten kann und welcher trotzdem ein hohes Maß an Interaktivität durch entsprechend schnelle Berechnungs- und Visualisierungsmethoden erlaubt. Bereits in dieser kurzen Beschreibung wird der enge Bezug dieser Anwendung zum in Kapitel 3.3 beschriebenen Benutzerkontext offensichtlich. Zur Erreichung der vorgegebenen Ziele sind hauptsächlich Eingriffe in den Datengenerierungsprozess, d. h. eine Anpassung an den Datenkontext erforderlich. Hierzu wird die sonst übliche zeitaufwendige numerische Simulation durch einen regelbasierten Morphingansatz ersetzt, welcher einen gegebenen Basisbekleidungsdatensatz einer vorgegebenen Größe effizient in ein benutzerindividuelles Bekleidungsstück transformiert.

Die hier aufgeführten Forschungsarbeiten sind Bestandteil mehrerer wissenschaftlicher Veröffentlichungen ([25]-[27]) auf internationalen Konferenzen und wurden innerhalb des Verbundprojektes Virtual Try-On [121] vom Bundesministerium für Bildung und Forschung (BMBF) unter dem Förderkennzeichen 01IRA01H gefördert.

5.1 Existierende Lösungen

Bisherige Ansätze für die virtuelle Anprobe von Bekleidung sowohl im realen Shop als auch im Internet haben nicht zum erhofften kommerziellen Erfolg geführt. Ein wesentlicher Grund hierfür liegt in der nicht vorhandenen Identifikationsmöglichkeit des Kunden mit dem ihm präsentierten Modell, an dessen Körper

er die Passform eines Bekleidungsstückes beurteilen und anschließend eine Kaufentscheidung tätigen soll. Bestehende Systeme basieren entweder auf rein zweidimensionalen Schablonen oder vereinfachten Computer-Mannequins, in denen ein Kunde sich nur schwer wieder erkennen kann. Ähnliche technologische Defizite bestehen ebenfalls in der kundenadäquaten Visualisierung des individuellen Bekleidungswunsches. Bestehende Systeme basieren entweder auf rein zweidimensionalen Darstellungen oder arbeiten mit vereinfachten Modellen zur Textilsimulation, die zwar „Zeichentrickqualität“ in der Darstellung liefern, jedoch nichts über die Passform eines Kleidungsstückes aussagen. Gerade beim Vertrieb über Kataloge oder das Internet führen deshalb hohe Rücklaufquoten aufgrund mangelnder Passform zu erheblichen finanziellen Einbußen im Handel und in der Produktion.

In den beiden folgenden Abschnitten werden existierende Shop-Umgebungen sowie existierende Techniken zur Simulation von Stoffen und Kleidung vorgestellt und im Hinblick auf den Einsatz im E-Commerce im Bereich von Konfektions- und Maßbekleidung bewertet.

5.1.1 Virtuelle Shop-Umgebungen

Bei den virtuellen Shop-Umgebungen lassen sich prinzipbedingt zwei Ansätze unterscheiden. Während in einigen Umgebungen rein zweidimensionale Verfahren basierend auf Fotografien des Kunden zum Einsatz kommen, werden bei anderen Firmen 3D-Techniken basierend auf vorgegebenen Models mit nur geringen Kundenidentifikationsmöglichkeiten verwendet. Als Beispiel für die 2D-Verfahren wird im Folgenden kurz auf den Otto Versand eingegangen, Land’s End repräsentiert dagegen eine 3D-Technik.

Zur Nutzung des vom Otto Versand (http://www.otto.de) zur Verfügung gestellten Angebotes muss der Kunde ein gewisse Voraussetzungen erfüllendes Bild von sich (die Aufnahme sollte frontal mit leicht vom Körper abstehenden Armen gemacht werden) an den Otto-Versand einschicken. Alternativ kann auch aus jeweils drei männlichen und weiblichen Modellen unterschiedlicher Statur ausgewählt werden. Anschließend sucht man die gewünschten Kleidungsstücke zur Anprobe aus. Diese werden nun entsprechend der Form des zugehörigen Körperteils geformt und verdeckend vor diesen gesetzt. Die Kleidungsstücke wirken dadurch jedoch wie „aufgeklebt“. Die Analyse der Passform wird rein anhand des zweidimensionalen Bildes ausgeführt. Dies bietet zwar den Vorteil einer sehr schnellen Verarbeitung, so dass der Kunde analog einem normalen Katalog durch die verschiedenen Kleidungsstücke blättern kann, beinhaltet aber natürlich auch viele Nachteile.

Abbildung 5-1: Virtuelle Anprobe im Otto Online-Shop.

Die mit diesem reinen 2D-Verfahren erzeugten Bilder wirken gerade in der angebotenen Vergrößerung größtenteils sehr unrealistisch. Zudem geht durch den Zuschnitt auf die exakten Körperformen der Personen jegliche Information über das Fallen der Kleidung verloren. Abbildung 5-1 zeigt die Ergebnisse einer solchen virtuellen Anprobe. Es lässt sich leicht erkennen, dass die erzeugten Bilder mit der Realität wenig zu tun haben. Die in Wirklichkeit relativ gerade und locker fallende Bluse aus dem Katalog (2. Bild von links) wirkt in der Anprobe (1. und 2. Bild von rechts) hauteng anliegend, so dass dem Betrachter ein völlig falscher Eindruck vermittelt wird. Auch werden unterschiedliche Größen nicht korrekt wiedergegeben. Die Bundweite einer Hose wird beispielsweise immer passend gemacht und Detailänderungen wie die Kürzung der Beinlänge einer Hose sind erst gar nicht möglich. Gerade die unverzichtbare Entscheidungshilfe bei der Auswahl zwischen zwei benachbarten Konfektionsgrößen kann durch den 2D-Ansatz nicht geleistet werden, da eine solche Aussage mit reinen 2D-Methoden nicht getätigt werden kann. Es ist lediglich eine Aussage darüber möglich, ob die ausgewählte Farbe und das Muster prinzipiell gefällt.

Im Vergleich zum gerade beschriebenen 2D-Ansatz setzt die amerikanische Handelskette Land's End (http://www.landsend.com) den 3D-Produktkonfigurator *My Virtual Model* ein. Bei diesem Verfahren muss der Kunde zunächst sein „persönliches" Modell kreieren. Hierzu trifft er bei einer Anzahl vorgegebener Merkmale (Geschlecht, Figurtyp, Größe, Gewicht, Gesicht, Haarfarbe usw.) eine Auswahl aus den entsprechenden Vorgaben (siehe Abbildung 5-2).

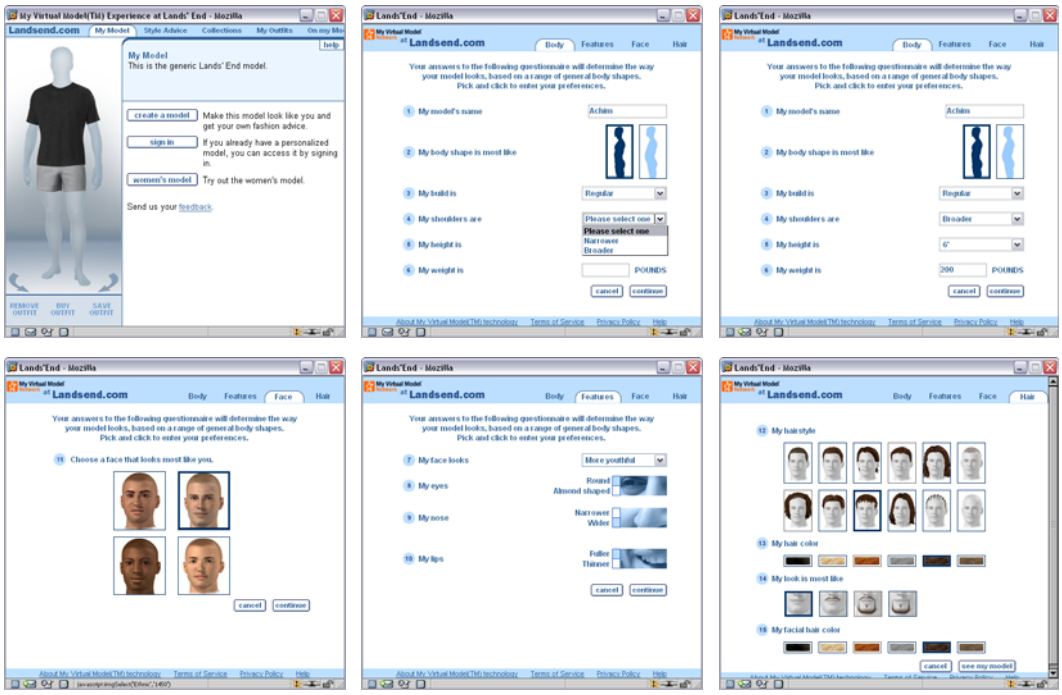

Abbildung 5-2: Land's End: Definition des virtuellen Modells.

Die gebotenen Möglichkeiten zur Individualisierung der Figurine sind offensichtlich sehr begrenzt, zumal sich die 3D-Avatare bei der Konfiguration nur minimal ändern. Entsprechend ist die Ähnlichkeit des Kunden mit der erzeugten Figurine (siehe Abbildung 5-3) nur minimal und damit eine Identifikation des Kundens mit derselben sowie Aussagen über die ästhetische Qualität eines Bekleidungsstückes in Verbindung mit dem Kundenkörper (Haut- und Haarfarbe, Typ) kaum möglich.

Abbildung 5-3: Aus den Parametern der Abbildung 5-2 generierte Figurine des Autors.

Nach Erzeugung der Figurine kann der Online-Kunde die virtuelle Puppe mit vorgefertigten Bekleidungsteilen „bekleiden". Die Bekleidungsteile sind dabei vorgefertigte, einfache CAD-Modelle und vermitteln keinen realistischen Eindruck vom

späteren Aussehen am Kunden. Die visualisierten Bekleidungsstücke wirken sehr unrealistisch, eine Passformaussage ist nicht möglich: Unabhängig von der ausgewählten Größe suggeriert das System dem Benutzer, dass die ausgewählte Bekleidung passt. Abbildung 5-4 gibt einen Eindruck von der virtuellen Anprobe von Jeans und Pulli bzw. Hose und Jackett.

Abbildung 5-4: Land's End: Virtuelle Anprobe von Bekleidungsstücken.

Zusammenfassend lässt sich sagen, dass die auf dem Markt befindlichen Systeme zwar erste Ansätze zur virtuellen Anprobe liefern, jedoch nur relativ unrealistische oder gar verfälschende Darstellungen bieten. Zudem fehlt gerade der eigentlich wichtigste Zielfaktor solcher Systeme, nämlich die Individualisierung und die damit eng verbundenen Identifikationsmöglichkeiten des Kunden. Hierdurch werden die angebotenen Werkzeuge für den Kunden eher zu einem Spielzeug als zu einer wirklichen Entscheidungshilfe – entsprechend fehlt den existierenden Shopumgebungen die nötige Kundenakzeptanz. Einen Beleg hierfür liefert das amerikanische Versandhaus Lane Bryant, welches die virtuelle Anprobe mittels des *My Virtual Model*-Systems nach Kundenumfragen mittlerweile eingestellt hat. Zur Begründung heißt es auf deren Web-Seiten (http://www.LaneBryant.com): „*We continually ask for feedback from our customers. This allows us to improve your online shopping experience. A recent assessment of the 3D model technology ME@LB (provided by My Virtual Model) revealed that most of our customers no longer use this feature. After much thought, we decided to remove ME@LB from the site [...]*“.

5.1.2 Simulation von Stoffen und Kleidung

Eine weitere, sich deutlich von den obigen Online-Shop-Ansätzen abhebende Möglichkeit zur Schaffung eines Individualkataloges besteht in der Simulation von Stoffen und Kleidung. Bis vor wenigen Jahren musste sich hier die Computergrafik mit der Modellierung starrer Objekte begnügen. Die Entwicklung neuer Algorithmen und stark verbesserter Grafikhardware erlaubten dann aber auch die Modellierung äußerst flexibler Objekte, deren Verhalten durch die Elastizitätsthe-

orie beschrieben wird. Stoffe und Kleidung sind eine Klasse solcher flexibler Objekte, insbesondere sind sie besonders starken Deformationen ausgesetzt. Dabei muss beachtet werden, dass ein Stoff kein kontinuierliches Medium ist, sondern eine komplexe Mikrostruktur besitzt. Aus diesem Grund ist eine Simulation des Verhaltens von Stoffen ein äußerst komplexer Vorgang.

Die existierenden Visualisierungs- und Modellierungstechniken für Stoffe lassen sich in drei Bereiche unterteilen.

a) *Geometrische Techniken*: Hier wird die Form des Stoffes ohne physikalische Modelle berechnet, z. B. durch B-Spline- oder polygonale Flächen.

b) *Physikalische Techniken*: Sie berechnen die Form der Stoffe unter Beachtung eines physikalische Modells, das von Ansatz zu Ansatz variiert und unterschiedliche Eigenschaften (z. B. Gravitation, Luftwiderstand, Bending, Buckling usw.) und Genauigkeit besitzt.

c) *Hybride Techniken*: Hier wird versucht, die Vorteile von geometrischen und physikalischen Techniken miteinander zu kombinieren.

Im Folgenden sollen beispielhaft die bekanntesten Verfahren aus den drei Bereichen vorgestellt und abschließend im Hinblick auf die Aufgabenstellung „interaktiver Individual-Bekleidungskatalog“ bewertet werden.

5.1.2.1 Geometrische Verfahren

Geometrische Modelle betrachten keine physikalischen Eigenschaften der Stoffe, sie konzentrieren sich mehr auf das Aussehen, insbesondere auf Falten, die sie mit geometrischen Gleichungen beschreiben. Geometrische Techniken erfordern ein gewisses Maß an Benutzerinteraktion; man kann sie daher in gewisser Weise als ein speziell angepasstes Zeichenwerkzeug betrachten. Im Rahmen dieser Arbeit sollen zwei Methoden näher beschrieben werden: die Arbeit von J. Weil und das interaktive Bekleidungsdesign.

J. Weil [128] nimmt für seinen Ansatz an, dass der Stoff quadratisch ist und betrachtet ihn als ein Gitter von dreidimensionalen Punkten, das Gitter selbst als zweidimensionales Koordinatensystem. Der Stoff wird an ausgewählten Punkten in Raum befestigt. Weil beschreibt zwei Bearbeitungsschritte, denen ein darzustellendes Stoffobjekt unterzogen wird. Im ersten Schritt wird eine Annäherung an die Oberfläche innerhalb der durch die Befestigungspunkte definierten konvexen Hülle vorgenommen. Einige natürliche Eigenschaften des Stoffes werden in diesem Teil der Berechnung ignoriert. Deshalb sind zum Beispiel die Falten, die der Stoff üblicherweise wirft, nach dem ersten Bearbeitungsschritt noch nicht zu sehen. Den Eigenschaften des Stoffes wird im zweiten Bearbeitungsschritt Rechnung getragen, der einen iterativen Relaxationsprozess beinhaltet. Die Relaxation eines Punktes auf der Oberfläche wird so lange wiederholt, bis die maximale Ver-

schiebung unter eine vordefinierte Toleranzgrenze fällt. Die Oberflächenannäherung dient dazu, das darzustellende Objekt zunächst grob in seine Form zu bringen, um dann im zweiten Bearbeitungsschritt in Abhängigkeit von den Materialeigenschaften die genaue Form zu bestimmen.

Für ein Stück Stoff aus gewebten Fäden kann man die Positionierung der Punkte entlang eines einzelnen Fadens durch Untersuchung der Kurve, die ein solcher Faden beschreibt, gewinnen. Diese Kurve wird „Catenary“ genannt und ist durch eine Gleichung der Form

$$y = \cosh\left(\frac{x}{a}\right)$$

gegeben. Dabei ist x eine gegebene x-Position, y die zu berechnende, zugehörige y-Position und a eine ebenfalls zu bestimmende Konstante (für eine detaillierte Beschreibung sowie die Herleitung der Gleichung siehe [128]). Indem man für jedes Paar von Befestigungspunkten die zugehörige Catenary entlangläuft, kann man also die Gitterpunkte auf den Linien zwischen den Punkten festlegen. Falls sich zwei Catenaries überschneiden, wird eine neue Kurve errechnet, die die Form des Stoffes genauer festlegt. Der Rest der groben Form des Stoffes wird durch wiederholte Unterteilung der Dreiecke, die durch die sich überlagernden Catenaries gebildet werden, festgelegt. Jedes Dreieck wird dabei separat betrachtet und immer wieder weiter unterteilt, bis jeder Gitterpunkt im Inneren des Dreiecks positioniert wurde. Nachdem alle Dreiecke so bearbeitet worden sind, ist die Oberfläche innerhalb der Befestigungspunkte angenähert. Danach wird in der Relaxationsphase die endgültige Form des Stoffes festgelegt, wobei hier den physikalischen Eigenschaften des Stoffes Rechnung getragen wird.

Die Berechnungen, die in diesem Abschnitt beschrieben wurden, ermöglichen eine halbwegs realistische Darstellung von Stoffobjekten. Viele der beschriebenen Probleme werden nicht exakt gelöst, jedoch erreichen die Annäherungen akzeptable Ergebnisse. Hier wurde ein sehr spezielles Problem behandelt: ein Stück Stoff ist an bestimmten Punkten fixiert. Weil ist der Ansicht, dass seine Algorithmen, auch auf andere, allgemeinere Probleme erweitert werden können. Weitere Verbesserungen könnten im Zeit- und Speicherbedarf möglich sein. Das zentrale Problem seines Ansatzes besteht allerdings darin, dass nur hängender Stoff modelliert worden ist, was z. B. für eine Bekleidungssimulation nur sehr eingeschränkt einsetzbar ist.

Im Vergleich zum Ansatz von Weil ist das interaktive Kleidungsdesign deutlich mehr an der Praxis orientiert und beschäftigen sich vor allem mit der Untersuchung, wie die traditionelle Vorgehensweise beim Entwurf von Kleidungsstücken durch computergrafische Methoden unterstützt werden kann. Die vorgestellten

Methoden basieren dabei im Wesentlichen auf der Arbeit von B. K. Hinds und J. McCartney [54].

Der Designprozess für ein Kleidungsstück beinhaltet mehrere verschiedene Schritte. Die anfängliche Skizze, die die Kreation des Designers darstellt, muss in ein Schnittmuster überführt werden, das, wenn es zu dem gesamten Kleidungsstück zusammengesetzt wird, das gewünschte Ergebnis liefert. Um dieses Ergebnis zu erzielen, gibt es einen Prozess, der die Schnittmuster in kontrollierter Weise verändert, um Passform oder verschiedene Materialien zu testen. Dabei wird immer nur eine Größe betrachtet. Wenn mit der Produktion des Kleidungsstückes begonnen wird, werden verschiedene Größenregeln benützt, um festzustellen, wie das Originalmuster verändert werden muss, um der Bandbreite der benötigten Größen gerecht zu werden.

Der bisherige Einsatz von Computern in diesem Gebiet beschränkte sich auf den Einsatz von Skizzensystemen; die Untersuchung der Passform wurde bisher immer durch Erfahrungswerte oder durch die Anprobe an einer Puppe vorgenommen.

Um in diesen Bereich vorzudringen, ist es nötig, ein Kleidungsstück so zu betrachten, dass es eine dreidimensionale Anordnung von Oberflächenstücken besitzt, die durch mathematische Modelle beschrieben werden können. Sind solche Oberflächen mit dem Rechner erstellt worden, erhält man mit einer 3D zu 2D Abbildung die Schnittmuster. Die Voraussetzung dafür ist, dass ein Kleidungsstück, welches aus einzelnen Teilen besteht, unter genauer Beachtung der Passform und des drapierenden Verhaltens definiert werden kann, dass eine ausgewählte Textur auf die Oberfläche aufgebracht werden kann und dass die resultierenden Schnittmuster definiert werden können, um eine Anordnung und Anpassung so schnell wie möglich sichtbar werden zu lassen. Ein Designwerkzeug dieser Art erlaubt nicht nur die Visualisierung der Stoffe, sondern ersetzt auch den Prozess der Erstellung der verschiedenen Stilmuster.

Ein auf dem obigen Ansatz basierendes Designwerkzeug wurde von Hinds und McCartney in dem diesem Abschnitt zugrunde liegenden Paper präsentiert. Um Daten für ein Computermodell zu erhalten, wurde in diesem Fall eine Schneiderpuppe digitalisiert und ein Array mit Radiuswerten erstellt, um eine Körperrepräsentation zu erhalten. Mit zusätzlichen Kontrollpunkten stellen die Oberflächenpunkte in diesem Array ein Kontrollpunktgitter für eine B-Spline-Oberflächeninterpolation dar. Durch Interpolation zwischen den gemessenen Punkten kann jeder beliebige Punkt auf der 3D-Körperform angesprochen werden.

Die Erstellung der Kleidungsstücke basiert auf 4 Forderungen:

i) Es sollte die Freiheit bestehen, eine beliebige Anzahl von Kanten für ein Kleidungsstück zu wählen. Bei Hinds kann sich diese Zahl zwischen zwei und zehn bewegen.

ii) Es sollte eine lokale C°-Stetigkeit herrschen, wenn sich zwei Teile eines Kleidungsstückes treffen.

iii) Die interaktive Erstellung der Teile sollte sowohl auf dem Körpermodell als auch auf Cursor-Interaktion basieren.

iv) Es sollte möglich sein, Erweiterungen wie drapierendes Verhalten und Falten in die Kleidungsstücke einzufügen.

Die ersten drei Forderungen konnten von Hind und McCartney's Algorithmen direkt erfüllt werden. Dabei wählten sie eine duale Darstellung zur Definition der Kanten. Während diese unter Benutzung eines normalen kartesischen Koordinatensystems visualisiert werden, existiert eine zweite Form, in der jede Krümmung durch eine parametrische Kurve in den Dimensionen *ijk* dargestellt wird. Dabei sind *i* und *j* die Höhen- bzw. Winkelindizes und *k* der Offset entlang der Normalen zwischen einem Körperpunkt und einem beliebigen anderen Punkt im 3D-Raum. Danach konzentrierten sich Hinds und McCartney darauf, die vierte Forderung zu erfüllen. Es wurde möglich Falten dadurch zu generieren, dass der Offset eines Punktes entlang einer Normalen durch eine harmonische Funktion überlagert wird. Dazu wird eine sinusodiale Funktion benutzt, die in einem Punkt *P* ihre maximale Amplitude besitzt und nach außen auf Null abfällt. Eine Falte kann jetzt also dadurch erzeugt werden, dass der normale Offset *k* eines Punktes durch diese Funktion verändert wird.

Zusammenfassend kann man sagen, dass Hinds und McCartney ihren selbst gesetzten Forderungen gerecht werden. Die Verbesserung, die das System gegenüber der herkömmlichen Vorgehensweise bietet, ist die Tatsache, dass der Designprozess unabhängig von der Konfektionsgröße wird. Durch Skalieren des Entwurfs kann eine große Bandbreite an Größen abgedeckt werden. Der Nachteil des Systems liegt in der Bedienbarkeit, da sehr viel Interaktion und Kenntnisse über computergrafische Zusammenhänge notwendig sind, da z. B. Befestigungspunkte sinnvoll gewählt werden müssen, um das System zu bedienen. Die geometrische Vorgehensweise bietet einerseits die Möglichkeit der Interaktion, andererseits kann so das drapierende Verhalten nur sehr eingeschränkt modelliert werden. Hierin liegt der Hauptnachteil des Systems. Da es dafür gedacht ist, den Prozess der Bestimmung der Passform zu unterstützen bzw. zu ersetzen, ist es notwendig, dass hier möglichst exakte Voraussagen getroffen werden können. Eine geometrische Vorgehensweise, die das Hauptaugenmerk auf die Optik richtet, wird hier nicht ausreichend sein.

5.1.2.2 Physikalisch basierte Verfahren

Um das dynamische Verhalten der Stoffe bei Animationen realistisch zu modellieren, reicht es nicht aus, nur die augenscheinliche Optik bzw. das Aussehen des Ergebnisses als Grundlage für ein Modell zu wählen. Man muss vielmehr die physikalischen Eigenschaften des Stoffes beachten. Die Herausforderung bei einem physikalischen Modell liegt einerseits in der Konstruktion des Modells und andererseits in der Berechnung der endgültigen Form des Stoffes durch geeignete und schnelle numerische Methoden. Beispiele dafür sind das Lösen von Differentialgleichungen oder die Anwendung von stochastischen Methoden, um die minimale Energie eines Punktes der Stoffoberfläche zu berechnen.

D. Breen wählte für seine Arbeit einen physikalischen Ansatz. Dabei legte er besonderen Wert darauf, dass verschiedene Arten von Stoffen (z. B. Baumwolle, Wolle, Synthetik) möglichst realistisch und entsprechend ihren tatsächlichen Eigenschaften modelliert werden können und auch voneinander unterschieden werden. Weiterhin entwickelte er eine Möglichkeit, Gleichungen zur Beschreibung der Eigenschaften von Stoffen aus empirischen Daten, die etwa aus einer Materialuntersuchung gewonnen wurden, zu erzeugen.

Breens theoretisches partikelbasiertes Modell ([11],[12]) geht das Problem der Modellierung der 3D-Struktur eines fallenden Stoffes auf eine neue Art und Weise an. Das Modell basiert auf der Idee, dass ein Stoff kein kontinuierliches Medium ist, sondern eine komplexe mechanische Struktur besitzt. Es ist dabei aber weder wünschenswert noch von der Rechenzeit her praktikabel, alle Details der Fadenstruktur eines gewebten Stoffes nachzubilden. Es ist daher nötig, einen Kompromiss zu finden, der einerseits eine ausreichend schnelle Berechnung ermöglicht und andererseits die wichtigsten Wechselwirkungen zwischen den einzelnen Bereichen auf der Stoffoberfläche erfasst.

Breen modelliert den Stoff als eine Menge von Partikeln. Diese Partikel repräsentieren die Schnittpunkte von Längs- und Querfäden in einem ebenen Stück Stoff. Einige wichtige mechanische Eigenschaften eines gewebten Stoffes treten an diesen Schnittpunkten auf. Das sind beispielsweise die Schwerkraft oder die Tatsache, dass ein Stück Stoff nicht durch einen Gegenstand hindurch fallen kann, wenn er auf ihn auftrifft. Andere Wechselwirkungen, die nicht an den Schnittpunkten auftreten (Streckungen innerhalb der Fäden oder Krümmungen außerhalb der Stoffebene), können ebenfalls in das Modell integriert werden.

In seinem Modell repräsentiert Breen die verschiedenen Gegebenheiten und Wechselwirkungen, die auf der Fadenebene auftreten mit Energiefunktionen, welche einfache geometrische Beziehungen zwischen Partikeln innerhalb einer lokalen Umgebung abbilden. Die Energiefunktionen fassen fünf grundlegende in der

Fadenebene auftretende Eigenschaften zusammen. Jede Eigenschaft ist durch einen Term in der Gesamtenergiegleichung berücksichtigt:

$$E_{total,i} = E_{repel,i} + E_{stretch,i} + E_{bend,i} + E_{trellis,i} + E_{gravity,i}$$

Dabei beschreibt:

- $E_{total,i}$ die totale Energie eines Partikels i,
- $E_{repel,i}$ eine künstliche Abstoßungsenergie, die die Partikel untereinander auf einer minimalen Distanz hält,
- $E_{stretch,i}$ diejenige Energiefunktion, die jeden Partikel mit seinen vier Nachbarn in der Ebene verbindet und die Dehnung des Stoffes modelliert,
- $E_{bend,i}$ die Energie für Fäden, die aus der lokalen Ebene herausbewegt werden,
- $E_{trellis,i}$ die Energie für Fäden, die innerhalb der lokalen Ebene (ähnlich einer Scherung) bewegt werden und
- $E_{gravity,i}$ die Gravitationsenergie eines Partikels in Abhängigkeit seiner Masse.

Die spezifischen Energiefunktionen, die Breen anfangs benutzt hat, waren im Prinzip gute Rateergebnisse. Er leitete diese her, indem er passende Randbedingungen feststellte und dann Funktionen wählte, die diese interpolieren. Er implementierte eine Simulation des Modells als einen 2-Phasen-Prozess, der auf diskreten Zeitschritten arbeitet. Die erste Phase der Simulation beschäftigt sich mit dem Gravitationseffekt und behandelt die Kollisionen zwischen dem Stoff und einem geometrischen Objekt, mit dem der Stoff kollidieren könnte. Die zweite Phase benutzt eine stochastische Energieminimierungstechnik, um Wechselwirkungen zwischen den Partikeln festzustellen und die Konfiguration in ein lokales Energieminimum zu überführen, um dann mit dem nächsten Zeitschritt fortzufahren. Ebenso wird eine stochastische Technik benutzt, um Pertubationen in das Partikelgitter einzubringen, um so eine natürlichere asymmetrische Konfiguration zu erreichen.

Obwohl Breen vernünftige Simulationsergebnisse erzielen konnte, gab es noch einige Dinge, die er über seine simulierten Stoffe nicht wusste. Sein Modell basierte nicht auf tatsächlichen physikalischen Größen und deren Einheiten. Deshalb kannte er die tatsächliche Größe der simulierten Stoffstücke nicht und das Modell konnte auch keinerlei mechanische Informationen liefern. Der größte Nachteil jedoch war, dass er unterschiedliche Stoffarten nicht zufrieden stellend simulieren konnte.

Um seinem Modell eine bessere physikalische Grundlage zu geben, entwickelte er eine Technik um Energiegleichungen aus den Daten des Kawabata Evaluation Systems (ein System zur empirischen Analyse von Materialien, das die Materialeigenschaften in so genannten Plots, d. h. Funktionen zur Beschreibung der Eigenschaften, ausgibt) herzuleiten. Indem er einen wesentlichen Teil seiner Energiegleichungen auf Kawabata Daten ausrichtete, wurde sein Modell physikalisch und quantitativ. Das Modell deckt insgesamt zwei wesentliche Aspekte der Stoffmodellierung ab:

- Es ist möglich, die Eigenschaften von Materialien auf einer niederen Ebene, etwa die Mikrostruktur betreffend, zu simulieren. Damit wird es möglich, verschiedene Stoffarten nachzubilden.
- Weiterhin ist es möglich, makroskopische geometrische Strukturen von drapierten Stoffen zu erzeugen, die einen Vergleich mit dem tatsächlichen drapierenden Verhalten von Stoffen nicht zu scheuen brauchen.

Neben den Vorteilen, die das Modell zweifelsohne besitzt, muss aber auch auf die Nachteile des Ansatzes von Breen hingewiesen werden. Das Modell modelliert die statischen Eigenschaften der Stoffe sehr gut, aber dem dynamischen Verhalten (Beschleunigung usw.), wie es zum Beispiel in einer Animation nötig wäre, wird keine Beachtung geschenkt. Der Hauptgrund liegt hierbei wohl nicht darin, dass sich solche Überlegungen nicht in das Modell integrieren ließen, sondern vielmehr an dem utopisch hohen Bedarf an Rechenzeit. Dieser bewegt sich in der Größenordnung von einer CPU-Woche auf einer IBM RS6000, um ein Standbild eines drapierten Stoffes zu erzeugen. Eine andere Implementierung von Anderson auf einen Superrechner konnte diese Zeit auf mehrere Stunden reduzieren, was für eine Animation aber immer noch keinen wesentlichen Fortschritt darstellt.

Insgesamt kann man sagen, dass dieses Modell sehr gute Resultate für die Modellierung des drapierenden Verhaltens von Stoffen liefert, obwohl es keine wirklich exakten Konfigurationen benutzt. Eine absolute Exaktheit ist in gewissem Sinne auch unmöglich, wenn man die chaotischen und stochastischen Eigenschaften eines Stoffes betrachtet, der über einen Gegenstand fällt. Man kann nicht erwarten, dass ein Stoffstück jedes Mal, wenn es fällt, sich immer gleich verhält; daher kann man ebenso nicht erwarten, dass eine stochastische Simulation überhaupt absolut exakte Vorhersagen treffen kann.

Viele nachfolgende Arbeiten auf diesem Gebiet basieren auf dem Modell von Breen und integrieren zum Beispiel die Berechnung von dynamischem Verhalten [22]. Weiterhin wurde das Modell auch verändert, um Luftwiderstand zu modellieren. Das resultierende Verfahren erlaubt zum Beispiel die Simulation von Effekten wie Luftwiderstand, Wind sowie Reibung auf der Oberfläche von Körpern und die Bewegung der Körper.

Einen gänzlich anderen Weg als Breen verfolgt das Thalmann-Team [124]. Deren Ansatz basiert auf einem von Terzopoulos [118] entwickelten Modell zur Deformation von Körpern, welches eigentlich nicht primär zur Stoffsimulation gedacht war. Es hat sich jedoch gezeigt, dass sich dieses Modell trotz einiger Probleme (z. B. sehr große Deformationen) grundsätzlich zur Stoffsimulation eignet.

5.1.2.3 Hybride Verfahren

Hybrid-Techniken versuchen die Vorteile von geometrischen und physikalischen Techniken zu kombinieren. Im Wesentlichen sollen die Berechnungen für ein physikalisches Modell durch geometrische Hilfstechniken beschleunigt werden.

Als Beispiel für hybride Verfahren sei der Ansatz von Taillefer [117] genannt, welcher in seiner Arbeit die Falten eines hängenden Stoffes modelliert, indem er sie in zwei Arten unterteilt: horizontale und vertikale Falten.

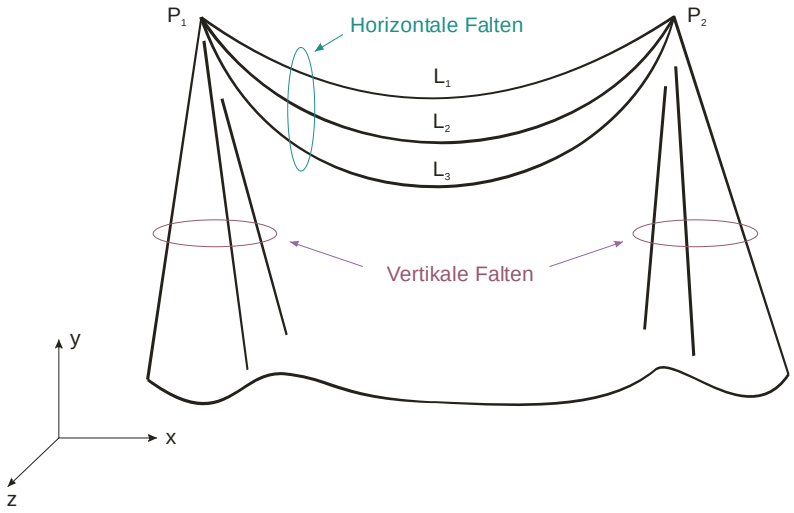

Abbildung 5-5: Horizontale und vertikale Falten.

Die horizontalen Falten werden durch Catenaries modelliert, wie es Weil bereits vorgeschlagen hat (dies ist also der geometrische Aspekt). Der Offset (*z*-Wert) jeder horizontalen Falte von der Ebene, die die Befestigungspunkte P_1 und P_2 enthält, wird dabei durch die folgende Gleichung berechnet:

$$z = z_{max} \frac{\operatorname{atan}\left(10 \frac{l - P_1P_2}{L}\right)}{\frac{\pi}{2}}$$

Die vertikalen Falten werden durch einen Relaxationsprozess ähnlich dem von Weil modelliert, mit dem Unterschied, dass mehr externe Bedingungen mit einbezogen werden. Diese sind Stretching, Bending, Gravitations- und Abstoßungsenergien (hierin findet sich also der physikalische Aspekt wieder). Um Rechenzeit zu sparen wird die Relaxation nur auf den Konturen der vertikalen Falten ausgeführt. Die Arbeit von Taillefer stellt also eine Erweiterung zu der von Weil dar und beschäftigt sich ebenso nur mit hängenden Stoffen.

5.1.2.4 Bewertung der Simulationstechniken

Geometrische Techniken versuchen die Lösung eines gegebenen Problems mit geeigneten Verfahren zur Lösungsannäherung zu finden. Das Ziel besteht nicht darin, eine exakte Lösung zu finden, es ist mehr das augenscheinliche Aussehen einer Konfiguration wichtig und weniger die physikalische Genauigkeit der Simulation. Die jeweiligen Verfahren werden sehr speziell an die Problemstellung angepasst. Hier liegt der Vorteil der geometrischen Techniken, denn sie sind durch die spezielle Ausrichtung auf eine Problemstellung sehr effizient und benötigten wenig Rechenzeit. Aber diese Spezialisierung stellt gleichermaßen auch einen großen Nachteil dar, denn die Verfahren sind so gut wie gar nicht auf andere Problemstellungen übertragbar. Externe Einwirkungen auf die Konfiguration wie Luftwiderstand oder Wind können durch die geometrischen Techniken nicht modelliert werden. Geometrische Techniken eignen sich daher besonders für Probleme, bei denen es um die Passform von Kleidung geht, denn hier ist ein spezialisierter Ansatz geeignet, der z. B. durch seine Schnelligkeit Interaktivität beim Kleidungsdesign gewährleistet. Weniger geeignet sind sie allerdings für Problemstellungen, bei denen eine realistische Animation von Stoffen oder Kleidung benötigt wird, da für diese Probleme insbesondere externe Faktoren (Wind, Luftwiderstand, Kollisionen usw.) maßgebend sind. Hier fehlt ihnen die Möglichkeit, physikalische Phänomene, die das Verhalten des Stoffes bestimmen, zu modellieren.

Bei physikalischen Techniken ist es normalerweise nötig, eine große Anzahl von Differentialgleichungen zu lösen oder viele Iterationsschritte durchzuführen, um ein Energieminimum zu finden. In der Tat ist das effiziente numerische Lösen von Gleichungen eines der Hauptprobleme. Die Techniken werden daher im Allgemeinen mittlere bis lange Rechenzeit benötigen. Die benötigte Zeit schwankt von einigen Minuten bis zu mehreren Tagen. Der Stoff selbst wird als Dreiecks- oder Viereckgitter mit einer endlichen Masse an den Schnittpunkten modelliert. Die Kräfte oder Energien eines Punktes werden unter Berücksichtigung der lokalen Nachbarn berechnet. Die Anordnung und Anzahl der berücksichtigten Nachbarn variiert von Technik zu Technik. Die Parameter, die in physikalischen Techniken benutzt werden, sind allgemein auch einem Benutzer ohne spezielleres Vorwissen verständlich, daher sind Kenntnisse über Interna der Verfahren zum Benutzen in der Regel nicht notwendig (z. B. ist jedem klar, was Gravitation ist; die genaue Umsetzung

kann dann das Modell intern autonom vornehmen). Das Hauptproblem der physikalischen Techniken besteht in der Kollisionsberechnung. Sie beansprucht schon für sehr einfache Konfigurationen (ein Tuch fällt über eine Kugel o. ä.) einen Großteil der Rechenzeit (bis zu 40% und mehr). Komplexere Umgebungen (z. B. Kollisionen mit einem Menschmodell) und komplexere Stoffkonfigurationen wie Jacken, Kleider, Hosen usw. erhöhen weiter die Notwendigkeit, Fortschritte auf dem Gebiet der Kollisionsberechnung zu machen. Zusammenfassend bieten die physikalischen Techniken eine sehr hohe Genauigkeit und die Möglichkeit, eine Vielzahl von Eigenschaften zu modellieren. Dafür müssen aber komplizierte Kollisionsberechnungen und generell lange Rechenzeiten in Kauf genommen werden.

Hybride Techniken versuchen die Vorteile von geometrischen und physikalischen Methoden zu verbinden. Im Wesentlichen wird versucht, physikalische Berechnungen durch vorherige geometrische Annäherungen zu beschleunigen. Ein Problem dabei ist, dass durch das Einbeziehen von geometrischen Methoden in ein physikalisches Modell dieses dadurch speziell wird, d. h. nur noch für eine spezielle Situation brauchbar ist. Weiterhin erreicht man durch die geometrischen Methoden keine wesentliche Beschleunigung, da auch Rechenzeit für den Aufbau und die Handhabung neuer Datenstrukturen benötigt wird. Falls die geometrischen Methoden nicht nur zur Annäherung benutzt werden, sondern wesentlicher Teil des Modells sind, wird dadurch die physikalische Genauigkeit wesentlich beeinflusst.

Zusammenfassend lässt sich sagen, dass sich zwar die physikalischen Verfahren aufgrund der erzeugbaren hochwertigen und realistischen Ergebnisse von Bekleidung für die Darstellung eignen würden, jedoch zugleich sehr zeitaufwendig in der Berechnung sind. Sie scheiden daher für den im Rahmen dieser Arbeit behandelten interaktiven Individualkatalog aus, da verursacht durch die Wartezeiten die nötige Kundenakzeptanz nicht erreicht werden könnte.

5.2 Virtuelle Bekleidungsanprobe

5.2.1 Allgemeine Bemerkungen

Das zentrale Ziel des Verbundvorhabens Virtual Try-On ist die Schaffung der technologischen Grundlagen für eine synergetische Verbindung des innovativen Angebots kundenindividueller Bekleidung (Maßkonfektion) mit dem Potential des E-Commerce unter Einsatz von VR-Methoden.

Dazu wurde eine lückenlose Prozesskette von der Virtualisierung des Kunden über die automatische Körpermaßerfassung mit Hilfe von 3D-Laserscannern bis hin zur photorealistischen dreidimensionalen Darstellung des virtuellen Kunden in dem

gewünschten Kleidungsstück umgesetzt. Die entstehende Plattform – der virtuelle Shop – soll für den Kunden sowohl von realen Boutiquen als auch über das Internet von zu Hause aus zugänglich sein. Die im Projektrahmen entwickelten Verfahren umfassen die hierfür notwendigen Basistechnologien und behandeln zwei konkrete Grundszenarien:

- Der virtuelle Shop in einer realen Boutique

  In der Boutique wird der Kunde mittels eines 3D-Body-Scanners eingescannt, dessen Körperoberfläche also dreidimensional erfasst. Auf diese Weise entsteht eine individuelle Figurine des Kunden – der virtuelle Kunde –, die als Grundlage für die automatische Ermittlung der Körpermaße des Kunden sowie der späteren Darstellung mit dem ausgesuchten Kleidungsstück dient. Im Anschluss an den Scanvorgang kann der Kunde in einem virtuellen Katalog blättern, in dem er die verschiedenen Kleidungsstücke an seiner Figurine betrachten kann. Ist der Kunde an einer Kombination besonders interessiert, so wird basierend auf seiner Auswahl und der berechneten Körpermaße ein virtueller Konstruktionsvorgang angestoßen und exakte, individuell auf die Figur des Kunden zugeschnittene CAD-Modelle der gewünschten Kleidungsstücke erstellt sowie die Figurine virtuell unter Zuhilfenahme spezieller Simulationstechniken bekleidet. Über einen virtuellen Spiegel, ein Display in Personengröße, kann der Kunde nun sein virtuelles Spiegelbild mit der neuen Kleidung von allen Seiten begutachten.

- Der virtuelle Shop im Internet

  Der Kunde kann von einem mit dem Internet verbundenen PC auf seine individuelle Figurine zugreifen. Hierzu muss er sich vorher in einer Boutique scannen lassen, damit ein entsprechend nutzbares 3D-Modell von ihm existiert. Analog dem virtuellen Katalog in der Boutique kann der Kunde hier die virtuelle Bekleidung an sich selbst begutachten. Die grafische Darstellung wird dabei an die vom jeweiligen Kundenrechner bereitgestellten Ressourcen angepasst.

Mit einer Umsetzung der aufgezeigten Prozesskette ergeben sich eine Reihe von Vorteilen gegenüber den in Abschnitt 5.1.1 beschriebenen Verfahren:

- Durch die Individualisierung der virtuellen Figurine und die fotorealistische Darstellung der virtuellen Anprobe wird eine optimale Identifizierung des Kunden mit dem Produkt ermöglicht.
- Der Kunde kann die Kleidung an sich selbst und nicht wie in einem Katalog an einem fotografierten Model beurteilen.
- Die physikalisch basierte Simulation der Passform am virtuellen Kunden ermöglicht qualitative Rückschlüsse auf die tatsächliche Passform. Insbesondere in Verbindung mit dem E-Commerce verringern sich hierdurch die Rücklaufquoten aufgrund mangelnder Passform.

- Probleme mit der Größe eines Kleidungsstückes können im Shop durch interaktive Korrektur behoben werden. Das modifizierte Kleidungsstück kann sofort betrachtet werden.
- Kleidung kann „on demand“ produziert werden. Lagerhaltung und unnötige Produktion von Kleidung wird auf ein Minimum reduziert, was zu einer immensen Kosteneinsparung im Textilbereich führt.

5.2.2 Interaktiver Individual-Katalog

Der in dieser Arbeit betrachtete Teilbereich des Virtual Try-On Projektes beschäftigt sich mit der Konzeption und Entwicklung eines interaktiven und individuellen Bekleidungskatalogs. Hierzu wurde eine intelligente, regelbasierte Morphingtechnik entwickelt, deren zentrales Anwendungsgebiet im auf realen Größen basierenden Morphing von Bekleidung liegt. Um die gesetzten Ziele erreichen zu können, verwendet die entwickelte Methode die Agententechnologie und eine flexible komponentenbasierte Visualisierungssystemarchitektur. Der interaktive Individualbekleidungskatalog benötigt zwei zentrale Eingaben:

- den virtuellen Kunden: 3D-Body-Scan des Kunden inklusive einer Zuordnung der hierbei gemessenen Maße zu geeigneten Bekleidungsgrößen und
- das virtuelle Bekleidungsstück: 3D-Referenzdaten für die Bekleidung, gewonnen durch 3D-Scans der Bekleidungsartikel in einer festen Größe an einer Schneiderpuppe.

Die Einsatzgebiete des Katalogs liegen dabei sowohl in der Boutique als auch im Internetshop. In der Boutique kann der Kunde in diesem blättern und sich beliebige Artikel in einer Vielzahl von Varianten und Stoffen aus dem Sortiment aussuchen. In die engere Auswahl gezogene Bekleidungsstücke können in der virtuellen Anprobe anprobiert und im virtuellen Spiegel in realer Größe dargestellt werden. Entsprechend kann der Kunde basierend auf seinen Daten von einem mit dem Internet verbundenen Rechner auf den virtuellen Shop und den hier ebenfalls integrierten interaktiven Individualkatalog zugreifen.

Eine wichtige Voraussetzung für den interaktiven Individualkatalog sind die aus den 3D-Scandaten gewonnenen Informationen. Daher wird im Folgenden Abschnitt auf die verwendete 3D-Body-Scanner-Technologie eingegangen.

5.2.3 3D-Body-Scanner

Im Rahmen der beschriebenen Applikation wird zur Datengewinnung der von der Wiesbadener Firma Vitronic Dr.-Ing. Stein Bildverarbeitungssysteme GmbH entwickelter 3D-Farb-Body-Scanner *VITUS* eingesetzt ([122],[123]). Primäres Ziel des Body-Scanners ist es, Menschen und andere lebende Objekte, die nicht belie-

big lange still stehen können, schnell und vollständig dreidimensional zu vermessen.

Der 3D-Scanvorgang ist dabei in der Durchführung sehr einfach: Man stellt sich in die Mitte des Messportals und auf Knopfdruck schalten sich die augensicheren Laserlichtquellen ein. Das System tastet nun die Konturen einer Person von Kopf bis Fuß ab. Wenige Sekunden später sind die Daten im Rechner gespeichert. Eine so genannte „3D-Punktewolke mit Farb-Overlay“, zusammengesetzt aus mehreren Millionen 3D-Messpunkten, ist entstanden. Mit millimetergenauer Präzision ist ein virtuelles Double der Person erschaffen.

Abbildung 5-6: Der *VITUS pro* Body-Scanner.

VITUS pro ist ein 360°-Scanner, welcher in lediglich 10 bis 20 Sekunden das Scanobjekt dreidimensional mit einer Auflösung von 1 bis 2 Millimetern erfasst. Der Scanner arbeitet mit einem robusten und industrieerprobten Lichtschnittprinzip zur 3D-Oberflächenrekonstruktion. Mit bis zu 16 Triangulationskameras kann ein Volumen von ca. 1,2 m x 0,8 m x 2,1 m gescannt werden. Optional kann durch den Einsatz von 6 Farbkameras jeder einzelne 3D-Punkt mit Farbtextur aufgenommen werden. Zeitgleich zur 3D-Datenerfassung wird dabei die Farbtextur der vermessenen Oberfläche mit Hilfe von Farbfotokameras aufgenommen. Für jeden 3D-Punkt wird die entsprechende Farbtextur aus dem Farbbild bestimmt und mit ihm verknüpft. Voraussetzung für die Berechnung ist, dass die Position der Farbkamera und der 3D-Oberfläche im Raum bekannt sind. Zunächst wird für jeden Bildpunkt (Pixel) der Farbkamera mit Hilfe einer Kalibrierung ein Sehstrahl

berechnet. Für jeden Sehstrahl der Farbkamera, der die 3D-Oberfläche trifft, wird anschließend die entsprechende Textur mit der 3D-Oberfläche verknüpft.

Neben dem äußerst leistungsfähigen Farbscanner *VITUS pro* wurde von Vitronic eine kleine, kompaktere Version des Ganzkörperscanners auf den Markt gebracht: *VITUS smart*. Durch einzelne technische Veränderungen hat der *VITUS smart* eine Größe erreicht, die beispielsweise eine Integration direkt in die Umkleidekabine eines Bekleidungsgeschäftes erlaubt. Im Gegensatz zum *VITUS pro* erfasst der *VITUS smart* nur die Geometrie des Scanobjektes, eine farbige Textur kann beim Scanprozess nicht aufgenommen werden.

Abbildung 5-7: Integration des *VITUS smart* im Verkaufsraum.

Alle *VITUS*-Scanner sind dreidimensionale berührungslose Scanner. Die Daten werden mit mehreren Triangulationskameras nach dem Messprinzip des Lichtschnittverfahrens aufgenommen.

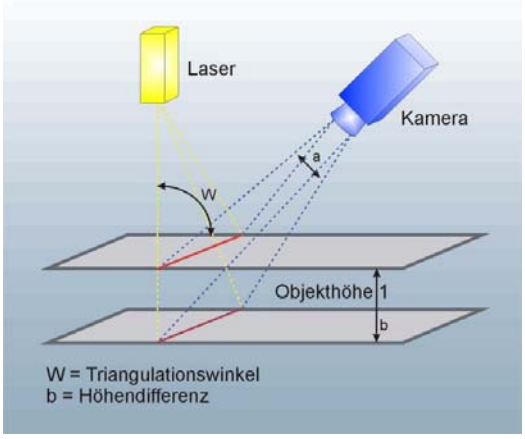

Abbildung 5-8: Prinzip des Lichtschnittverfahrens.

Diese Methode nutzt eine Laserlinie als Lichtquelle. Mit einer Videokamera wird die Position der Laserlinie auf dem Objekt von einem definierten Winkel her aufgenommen. Aufgrund des Triangulationswinkels (Winkel zwischen Kamera und Lichtquelle) und der festgelegten optischen Anordnung von Kamera und Beleuchtung kann die Position eines Profilschnitts in *x*- und *y*-Richtung errechnet werden. Um die Dreidimensionalität zu erreichen wird der Triangulationsmesser in bestimmten Schritten in *z*-Richtung bewegt. Mit der Information über den Abstand zwischen den Schritten wird die Kontur des Objektes Schnitt für Schnitt gescannt.

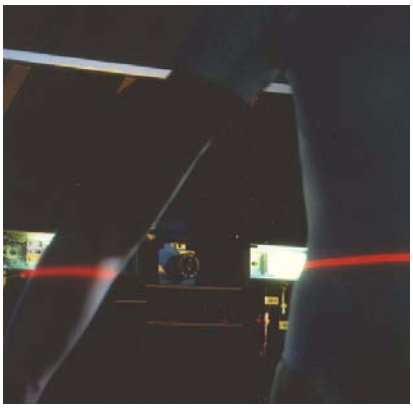

Abbildung 5-9: Scanvorgang.

Die technischen Daten der zurzeit produzierten Scanner können der folgenden Tabelle entnommen werden.

Scannertyp	***VITUS pro 8C***	***VITUS pro 16C***	***VITUS smart***
Triangulationskameras	8	16	8
Triangulationstyp	einfach	doppelt	
Gescannte Fläche (elliptisch) [m]	1,2 x 0,8		1,0 x 0,8
Gescannte Höhe [m]	2,1		2,04
Auflösung ($x \times y \times z$) [mm]	2 x 2 x 2		5 x 5 x 4
Scangeschwindigkeit (*z*-Auflösung: 2mm) [s]	21		n.v.
Scangeschwindigkeit (*z*-Auflösung: 4mm) [s]	11		
Textur (optional)	RGB		n.v.

Scannertyp	***VITUS pro 8C***	***VITUS pro 16C***	***VITUS smart***
Anschlusswert	230 V, 50 Hz, 2500 W		230 V, 50 Hz, 420 W
Abmessungen [m]	2,5 x 1,8 x 3,1		2,1 x 1,9 x 2,8
Gewicht [kg]	1200		120
Laserklasse	1 (augensicher)		

Tabelle 5-1: Technische Daten der *VITUS*-Scanner.

Die Anwendungsmöglichkeiten von 3D-Body-Scannern sind dabei vielfältig, ihr Einsatz erfolgt zum Beispiel:

- bei ergonomischer Forschung und Design: Designer der Automobilindustrie verwenden die 3D-Daten, um die Innenräume der Wagen unter ergonomischen Gesichtspunkten zu optimieren;
- in der anthropometrischen Forschung: Körperform und -maße eines repräsentativen Querschnitts der Bevölkerung werden exakt erfasst;
- bei der Vermessungen für Skulpturen: die Daten werden einer Fräsmaschine zugeführt, die aus einem Steinblock eine Skulptur als identisches Abbild fräst;
- bei der Vermessung für virtuelle Realitäten: in der Filmbranche werden virtuelle Schauspieler geschaffen und animiert. Hier könnten ganze Datenbanken von Schauspielern geschaffen, Bewegungsstudien erstellt und spezielle Haltungen digitalisiert und verwendet werden;
- im medizinischen Bereich: in der Orthopädie können ohne belastende Röntgenstrahlung die Form des Rückens oder anderer Gliedmaßen dreidimensional vermessen werden.

5.3 Intelligente Morphing-Technologie

Wie bereits erwähnt, liegt das Ziel der entwickelten Virtual Try-On Applikation in der Erzeugung von realistischer virtueller Bekleidung, ohne dazu jedoch zeitaufwendige physikalisch basierte Simulationstechniken anzuwenden. Die zentrale Idee liegt in der Ableitung eines Regelsystems, welches eine absolute und lokale Steuerung der Zwischenformen bei Anwendung des entwickelten Morphing-Verfahrens gewährleistet. Mit Hilfe dieser Technik kann dann jede gewünschte Größe eines gegebenen Bekleidungsmodells automatisch generiert werden. Nach einer kurzen Betrachtung existierender Morphing-Ansätze werden zunächst die in diesem Kontext gestellten Anforderungen definiert, aus denen dann die Regeln für

den Morphing-Agenten abgeleitet werden. Anschließend werden die entwickelten grundlegenden Techniken beschrieben, welche dann im darauf folgenden Abschnitt erweitert und verbessert werden.

5.3.1 Existierende Morphing-Ansätze

In den letzten Jahren haben sich Bildverarbeitungstechniken, insbesondere Morphing-Methoden, in der Film- und Unterhaltungsindustrie weit verbreitet. 2D-Morphing besteht im Wesentlichen aus der Transformation eines 2D-Bildes in ein anderes. Hierzu wird eine Funktion definiert, die vorzugebende Punkte des Startbildes auf Punkte des Zielbildes abbildet. Danach wird simultan die Farbe und die Position der zueinander gehörigen Punkte interpoliert, um Zwischenbilder zu erzeugen. Obwohl diese Technik für Filmanimationen recht nützlich ist, ist der Einsatzbereich relativ eingeschränkt. Um weitere Einsatzbereiche zu erschließen, müssen daher nicht nur 2D-Bilder sondern 3D-Objekte transformiert werden. Ein Vorteil der 3D-Verfahren liegt z. B. in der Möglichkeit, Objekte unabhängig von der Transformation animieren zu können.

3D-Morphing bedeutet im Wesentlichen die Simulation der Deformation von geometrischen Shapes. Die Art und Weise, wie die Deformation spezifiziert wird, ist für die Flexibilität und Benutzbarkeit des Deformations-Tools von entscheidender Bedeutung. Das Ziel besteht darin, ein allgemeines und intuitives Werkzeug zur Shape-Transformation bereitzustellen, ausgehend von einem Start- und einem Ziel-Shape (analog zu Start- und Ziel-Bild bei den 2D-Techniken). Die 3D-Morphing-Methoden können, abhängig von der Art der benutzten Information, in zwei Klassen unterteilt werden: volumetrische Ansätze und Boundary-Ansätze.

In volumetrischen Ansätzen wird wie der Name schon sagt nur volumetrische Information verwendet, um die Shape-Transformation von Objekten zu berechnen. Dabei gibt es keine Restriktion für die topologische Korrespondenz zwischen dem Start- und dem Zielshape; ein Torus kann also z. B. in eine Kugel verwandelt werden.
Kaul und Rossignac [66] definieren interne Punkte polyedrischer Zwischenobjekte durch lineare Interpolation basierend auf Minowski-Summen der internen Punkte der beiden ursprünglichen Polyeder. Ein deformierendes Objekt kann als parametrisierter Shape betrachtet werden, wobei der Parameter die Zeit ist. Ein solches Objekt wird als Parameterized Interpolation Polyhedron (PIP) bezeichnet, da es vollständig durch das Start- und Zielobjekt definiert wird und die Implementierung auf polyedrische Objekte beschränkt ist. Die Veränderung setzt sich aus einer Translation und einer Skalierung zusammen, die beide lineare Funktionen der Zeit sind. Die PIPs werden dadurch animiert, dass für jeden Zeitwert die Koordinaten der Vertices neu berechnet werden und die Faces einfach durch eine Tiefen-Puffer-Architektur gerendert werden. Die PIPs sind vollständig durch den

Anfangs- und Ziel-Shape, die beliebige Polyeder sein können, definiert. Die Polyeder müssen keine korrespondierenden Punkte besitzen und müssen auch nicht notwendig konvex sein. Das PIP verhält sich wie das gewichtete Minowski-Mittel der beiden Polyeder A und B: $(1-t)A + tB$ mit $t \in [0,1]$. Der Vorteil dieses Ansatzes liegt in seiner Robustheit und dem weitgehend automatischen Ablauf. Der entscheidende Nachteil ist die sehr eingeschränkte Interaktionsmöglichkeit, d. h. die Art und Weise, wie sich die Objekte ineinander verwandeln, ist so gut wie nicht zu kontrollieren.
Die von Huges [57] verwendete Technik basiert auf einer stetigen Interpolation zwischen den Fourier-Transformationen von zwei volumetrischen Modellen und der anschließenden Rücktransformation der Ergebnisse. Da eine lineare Interpolation zwischen den transformierten Datensätzen teilweise zu nicht zufrieden stellenden Ergebnissen führt, wird die Verarbeitung von hohen und niedrigen Frequenzen während des Interpolationsprozesses getrennt betrachtet.

Bei den Boundary-Ansätzen wird üblicherweise zur Transformation eines Shapes in einen anderen das Problem in zwei Schritte aufgeteilt. Der erste Schritt besteht darin, eine Abbildung zwischen jeweils einem Punkt auf der einen Oberfläche und einem Punkt auf der Oberfläche des anderen Objektes zu finden. Sobald diese Abhängigkeiten aufgestellt sind, wird im zweiten Schritt eine Sequenz von Zwischenmodellen erstellt, indem bei zusammengehörigen Punkten zwischen deren Positionen auf der einen und anderen Oberfläche interpoliert wird. Der erste Schritt wird als das Korrespondenzproblem bezeichnet, der zweite als Interpolationsproblem. Beide Probleme sind voneinander abhängig, da die Methode zur Lösung des Interpolationsproblems davon abhängig ist, wie die Abbildungsfunktion aufgestellt wurde. Bei den Boundary-Ansätzen werden die Objekte als polyedrische Objekte dargestellt.
Bethel und Uselton [10] betrachten das Korrespondenzproblem als Problem von Korrespondenzen in einem Graphen. Da die Geometrie der beiden Polyeder dabei meistens nicht beachtet wird, sind die visuellen Effekte während der Shape-Transformation nicht sehr intuitiv und nur schwer zu kontrollieren.
Bei dem Ansatz von Kent, Carlson und Parent [67] stellt man sich ausgehend von zwei Genus-0-Objekten vor, dass diese wie Luftballons aufgeblasen werden, bis sie kugelförmig werden. Jeder Punkt auf der Oberfläche der Objekte wird dabei eindeutig auf einen Punkt auf der Kugeloberfläche abgebildet. Betrachtet man nun jeweils diejenigen Punkte auf den beiden Oberflächen, die auf denselben Punkt der Kugeloberfläche abgebildet werden, als Paar, so ergibt sich eine Eins-zu-eins-Korrespondenz zwischen den Oberflächenpunkten der beiden Objekte. Diese Technik kann teilweise auch auf Nicht-Genus-0-Objekte ausgedehnt werden, wenn eine eindeutige Abbildung von der Oberfläche des Objektes auf die Einheitskugel gefunden werden kann. Der erste Schritt des Algorithmus besteht also in der Projektion der Topologie der beiden Objekte auf die Einheitskugel. Dann werden die beiden Topologien gemischt, indem die projizierten Faces der beiden

Modelle gegeneinander geclippt werden. Die gemischte Topologie wird dann auf die Oberfläche der Original-Modelle zurück projiziert. Diese Vorgehensweise generiert zwei neue Modelle, die den gleichen Shape und die gleiche Topologie haben wie die beiden Originalmodelle. Dies erlaubt nun im zweiten Schritt die Durchführung der Transformation zwischen den beiden Modellen durch lineare Interpolation zwischen den Koordinaten der jeweiligen Punktepaare.
Parent [91] benutzt einen rekursiven Subdivision-Prozess, um einen gemeinsamen Nachbarschaftsgraphen zu erzeugen. Bei dieser Methode kann der Benutzer explizit Korrespondenzen zwischen bestimmten Gebieten der Objekte festlegen. Der Nachbarschaftsgraph wird dann durch rekursive Unterteilung dieser Gebiete erstellt.
Lazarus und Verroust [74] betrachten das Korrespondenzproblem und die Interpolation gemeinsam, also nicht wie sonst üblich unabhängig voneinander. Dazu wird jedes Objekt gesampelt und in geeigneter Art und Weise parametrisiert. Die Korrespondenzen werden durch die Wahl der gleichen Diskretisierungsrate für beide Objekte automatisch erzeugt. Lazarus und Verroust konzentrieren sich auf Objekte, deren Geometrie sternförmig um eine Achse angeordnet ist. Für diese existiert eine 3D-Kurve innerhalb des Objektes - ihre Achse -, so dass jeder Punkt des Objektes eindeutig einem Punkt auf der Achse zugeordnet werden kann. Die Objekte können in der Regel in drei Teile zerlegt werden: zwei hemisphärische Teile, jeweils an den Enden des Objektes, und einen zylindrischen Teil, der die Punkte entlang der Achse des Objektes beinhaltet. Für jeden dieser Teile wird eine passende Parametrisierung, d. h. sphärisch bzw. zylindrisch, berechnet. Dabei wird im zylindrischen Koordinatensystem anstelle der z-Achse eine allgemeine 3D-Kurve verwendet. Unter Benutzung dieser Achsen wird ein Sampling der Objekte berechnet, wobei die jeweiligen natürlichen Parametrisierungen zugrunde liegen. Die Interpolation wird dann im Parameterraum vorgenommen.

Sowohl volumetrische als auch Boundary-Ansätze wenden die Transformation global an. Daher besitzt keiner dieser Ansätze lokale Kontrollmöglichkeiten bei den Zwischen-Shapes. Da die hier vorgestellte Virtual Try-On Applikation aber z. B. die Ärmel der Kleidungsstücke unabhängig vom Bekleidungs-Torso behandeln muss, ist gerade diese lokale Steuerung äußerst wichtig. Es lässt sich zusammenfassend daher feststellen, dass sich die existierenden Morphing-Techniken nicht für den vorliegenden Einsatzzweck eignen.

5.3.2 Voraussetzungen

Anstatt die virtuelle Bekleidung mit Techniken der Bekleidungssimulation zu erzeugen, kann ein Basis-Kleidungsmodell auch durch das 3D-Scannen von Kleidungsstücken, die z. B. an einer Schneiderpuppe angebracht sind, erzeugt werden. Hierdurch wird ein Modell mit sehr realistischer Faltenbildung erzeugt. Der Hauptnachteil dieser Technik ist die Notwendigkeit, jedes Kleidungsstück in jeder

Größe durch den 3D-Laser-Scanner vermessen zu lassen, wodurch ein immenser und nicht handlebarer Zeit- und Speicheraufwand entsteht. Im hier vorgestellten Morphing-Ansatz wird dagegen der Shape eines durch den 3D-Laser-Scanner erzeugten Kleidungsmodells intelligent transformiert, um Modelle für die verschiedenen Größen zu berechnen. Die gewünschten Kleidungsstücke müssen also nur in einer Grundgröße gescannt werden, alle anderen Größen werden durch den Morphing-Agenten berechnet.

Im Gegensatz zu den existierenden Morphing-Techniken muss eine absolute Kontrolle über die entstehenden Zwischen-Shapes möglich sein, um garantieren zu können, dass beispielsweise auf dem Transformationsweg von der Größe S zur Größe XXL die Größe L exakt erreicht wird. Es muss hierzu sicher gestellt sein, dass der Zwischen-Shape für die Größe L die exakten zugehörigen Einzelmaße dieser Bekleidungsgröße trifft und nicht nur annähernd ähnlich aussieht. Deshalb ist es notwendig, die Differenzbeträge zwischen zwei Größen (zum Beispiel zwischen L und XL) detailliert zu betrachten. Bekleidungsgrößen werden normalerweise durch einzelne Maße wie Halsumfang, Ärmellänge, Ärmeldurchmesser oder Rückenbreite definiert (siehe Tabelle 5-2).

	S	**M**	**L**	**XL**	**XXL**	**XXXL**
Kragenweite	37/38	39/40	41/42	43/44	45/46	47/48
Taillenumfang	110	116	124	134	144	154
Ärmellänge	88	89	90	91	92	93
Rückenlänge	82	82	82	85	90	90

Tabelle 5-2: Beispielhafte Maße für ein Hemd (in cm).

Am Beispiel des Hemds sieht man leicht, dass sich die Einzelmaße in der Regel nicht linear oder gleichmäßig ändern. So nimmt zwar die Ärmellänge beim Übergang von M nach L von 89 cm auf 90 cm zu, die Rückenlänge bleibt aber konstant bei 82 cm. Daher wird eine Menge von Regeln abgeleitet, die die einzelnen Änderungen beschreiben, die vom Morphing-Agenten vorgenommen werden müssen, wenn das Bekleidungsstück von einer Größe auf eine andere verändert wird. Bereits das einfache Beispiel eines Hemdes macht klar, dass dieser Prozess deutlich komplizierter ist, als nur eine Art Ein- oder Auszoomen anzuwenden.

Basierend auf diesen grundlegenden Überlegungen lassen sich mehrere konkrete Anforderungen an die zu entwickelnde Morphingmethode formulieren:

(A1) Möglichkeit des Morphings zwischen zwei vorhandenen Konfektionsgrößen zur Erzeugung dazwischen liegender Größen

(A2) Anwendbarkeit der Methode bei nur einer gegebenen Ausgangsgröße

(A3) Beachtung der exakten Maßvorgaben beim Morphing

(A4) Lokalität, d. h. einzelne Segmente der Kleidung müssen voneinander unabhängig behandelt werden können

(A5) Möglichkeit der Kopplung von Maßänderungen, z. B. zur Erzeugung von Konfektionsgrößen

(A6) Möglichkeit der Anpassung an verschiedene Stoffarten muss im Konzept beachtet werden

Aus den Anforderungen (A1) und (A2) können zwei verschiedene, entsprechend angepasste Morphing-Methoden, definiert werden:

Methode 1 mit Beachtung der Anforderung (A1): Als erster Schritt erfolgt eine Segmentierung des Kleidungsstückes, so dass einzelne Teile „sternförmig“ um eine Achse (dargestellt als Polygonzug) angeordnet werden können. Hierdurch wird zugleich die Forderung der Lokalität erfüllt. Nun werden die Segmente der beiden Ausgangsobjekte mit gleicher Diskretisierungsrate gesampelt, d. h. die erforderliche Korrespondenz wird hierdurch automatisch erzeugt. Das Sampling orientiert sich dabei an den geometrischen Grundformen der Objekte und ist z. B. beim Arm zylindrisch. Die Schnitte zwischen den einzelnen Segmenten werden als Singularitäten betrachtet. Die Interpolation zwischen den zwei Ausgangsobjekten erfolgt durch simultane Interpolation der Achsen und lokalen Koordinaten, wobei sich die Art und Weise der Interpolation aus bekleidungstechnischen Größentabellen ableitet. Der Vorteil dieser Methode liegt in der Möglichkeit, zwischen zwei Größen stufenlos überblenden zu können. Nachteilig ist jedoch, dass die Falten der beiden Ausgangsobjekte „gemischt“ werden, was in ungünstigen Fällen eine unnatürlich anmutende Faltenbildung zur Folge hat.

Methode 2 mit Beachtung der Anforderung (A2): Analog zur Methode 1 wird zunächst das Kleidungsstück segmentiert und die zugehörigen Achsen bestimmt. Der eigentliche Morphing-Prozess wird durch Verlängerung der Achsen bzw. durch Änderung des Umfanges bezüglich der Achsen durchgeführt, d. h. ausgehend von einer gegebenen Größe wird eine Deformation zur Erzeugung der korrekten Maße durchgeführt. Im Gegensatz zur ersten Methode erfolgt hier somit keine „Mischung der Falten“. Durch gezieltes Anwenden des Morphing-Algorithmus in unterschiedlichen Regionen ist zudem die Integration von Stoffparametern zur Unterscheidung von Stoffeigenschaften prinzipiell möglich, jedoch im Rahmen dieser Arbeit nicht untersucht worden. Nachteilig wirkt sich aus, dass Falten je nach Abstand der erzeugten Bekleidungsgröße zur Größe des Ausgangsobjektes im Vergleich zum Original zu groß oder zu klein wirken können. Diesem Nachteil kann jedoch durch die Verwendung verschiedener gescannter Basismodelle für das gleiche Bekleidungsstück entgegengewirkt werden, da hierdurch der maxi-

male Abstand zu den jeweils damit zu erzeugenden Größen verringert werden kann.

Im Vergleich zu den bekannten Morphing-Verfahren lassen sich die oben beschriebenen Methoden wie folgt bewerten: Die Vorteile des Morphings liegen in der Möglichkeit des stufenlosen Überblendens zwischen zwei Größen, dem äußerst intuitiven Ansatz über Achsen, der hohen Flexibilität sowie der Tatsache, dass der eigentliche Morph kaum Anforderungen an die Rechenleistung stellt (Anwendung reiner Vektoraddition). Nachteile liegen vor allem in der zum Teil aufwendigen Behandlung von Problemen an den Segmentanschlussstellen, zum Beispiel im Schulterbereich.

Als Alternative zu den beschriebenen Verfahren kann man sich auch FFDs (Free Form Deformation) vorstellen, deren Vorteile in einer flexibleren Deformation liegen. Die Qualität des Ergebnisses eines FFDs ist jedoch stark abhängig von der Wahl eines geeigneten Kontrollgitters (Positionierung, Größe), welches wiederum von Kleidungsstück und Größe abhängt. Eine Überblendung zwischen zwei Objekten ist hier nicht möglich, es können lediglich die Kontrollgitter ineinander überführt werden.

Aufgrund der beschriebenen Vor- und Nachteile wurde im Rahmen der vorliegenden Arbeit die Methode 2 entsprechend den unter (A2) bis (A6) gesetzten Anforderungen näher untersucht und umgesetzt. Die Entwicklung erfolgte dabei mehrstufig:

i) Low-Level-Funktionalitäten: Längen- und Weitenänderungen einzelner Teilsegmente (z. B. Oberarm, Unterarm usw.)

ii) Integration von Korrekturtermen, die eine automatische Translation angeschlossener Teile desselben Segments erlauben (d. h. dass z. B. bei Längenänderung des Oberarms die Position des Unterarms entsprechend korrigiert wird)

iii) Integration von Korrekturtermen, die eine automatische Translation anderer Segmente bezüglich den gemachten Änderungen erlauben. Hierdurch wird u. a. eine Durchdringung unterschiedlicher Segmente vermieden (z. B. müssen bei Erhöhung der Bauchweite die Arme entsprechend „nach außen“ verschoben werden).

iv) Integration von Korrekturtermen, die die automatische Erzeugung glatter Übergänge zwischen den gemorphten Segmenten erzielen.

v) Einführung von „Size Fit Constrains“, d. h. Umsetzung von Maßänderungen anhand von Größentabellen (hierdurch kann z. B. aus Größe 52 Größe 54 berechnet werden).

vi) Virtuelle Anprobe, d. h. die Haltung von Figurine und gemorphtem Kleidungsstück müssen in Einklang gebracht werden.

Wie bereits erwähnt, wird für die hier vorgestellte Technik ein reales Kleidungsstück in nur einer einzigen bekannten Größe mittels des 3D-Body-Scanners erfasst. Es stellt sich damit die Frage, wie man in diesem Fall überhaupt eine Morphing-Technik anwenden kann, da entsprechende Methoden wie im vorhergehenden Abschnitt beschrieben ja einen Anfangs- und einen End-Shape als Eingabe benötigen. Die grundlegende Idee des im Rahmen der Arbeit entwickelten Verfahrens liegt daher in der Anwendung von Deformationen auf den Shape, welche die gewünschte Größe mit entsprechenden Maßen erzeugen. Damit deformiert der vorgestellte Ansatz den Shape und morpht zwischen den einzelnen Größen. Der Vorteil dieses Ansatzes gegenüber der Interpolation zwischen zwei Shapes (ein Shape einer kleinen Größe und ein Shape einer großen Größe) ist, dass hier keine zusätzlichen Falten bei den Zwischen-Shapes erzeugt werden, wenn die Falten an verschiedenen Positionen bei Anfangs- und End-Shape auftreten (was in der Regel bei zwei verschiedenen Scans immer der Fall ist). Diese Eigenschaft führt zu einem realistischeren Aussehen des neu berechneten Kleidungsobjektes. Der beschriebene Vorgang lässt sich leicht erweitern: durch Hinzufügen von zusätzlichen Scans desselben Kleidungsstückes in anderen Größen kann die Genauigkeit der Darstellung nochmals erhöht werden, wenn von einem Agenten diejenige Grundgröße mit minimalstem Abstand zur gewünschten Größe ausgewählt wird.

5.3.3 Basis-Morphing-Technologie

Wie bereits in Abschnitt 5.3.2 beschrieben, müssen die einzelnen Maße eines Kleidungsstückes individuell änderbar sein. Die Morphing-Technik muss daher eine lokale Kontrolle über die einzelnen Teile des Bekleidungsstückes ermöglichen, so dass sich z. B. eine Änderung am Ärmel nicht auf den Taillenumfang auswirkt. Aus diesem Grund wird vor Anwendung des Morphing-Prozesses das Kleidungsstück segmentiert. Dabei lehnt sich die Segmentierung an die einzelnen Kleidungsteile an. Ein Hemd wird so beispielsweise u. a. in unteren und oberen Ärmel, linken und rechten Brustteil sowie linken und rechten Rückenteil unterteilt.

Nach Durchführung der Segmentierung wird für jeden Teil eine geeignete Parametrisierung berechnet (siehe Abbildung 5-10). Beispielsweise werden in der Schulterregion sphärische Koordinaten verwendet, wogegen beim Arm zylindrische Koordinaten zum Einsatz kommen.

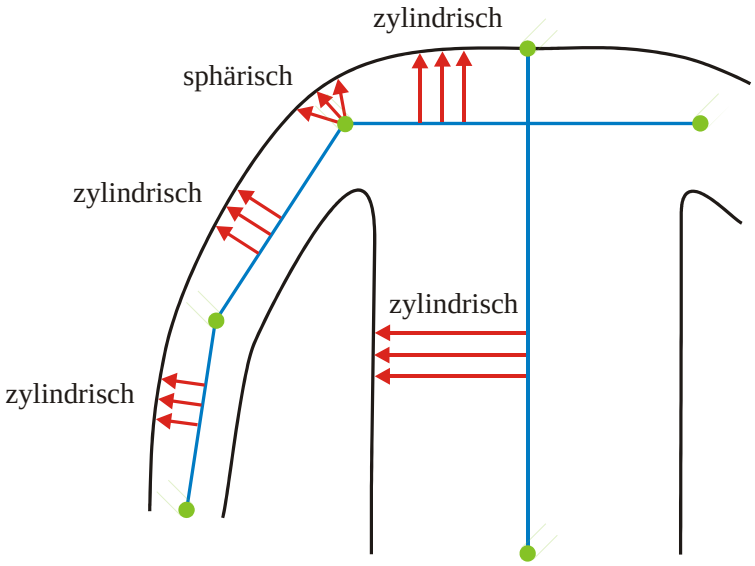

Abbildung 5-10: Segment-spezifische Parametrisierung.

Lässt man bei der zylindrischen Parametrisierung als Achse anstatt einer Geraden eine polygonale Kurve zu, so kann sichergestellt werden, dass die Achse z. B. völlig innerhalb des Ärmels liegt. Änderungen der Ärmellänge können nun sehr intuitiv durch Verlängerung oder Verkürzung der zugehörigen Zylinderachse durchgeführt werden. Entsprechend ergibt sich eine Weitenänderung des Ärmels durch eine Vergrößerung oder Verkleinerung des Abstandes der Ärmel-Vertices zur Achse. Führt man diese Methode nun für jedes Segment des Bekleidungsstückes aus, so definiert sich die Regel für den Gesamtmorph als Addition der einzelnen lokalen Deformationen:

$$M_{tot} = M_{sl} + M_{sw} + M_{ww} + \ldots + M_{tl}$$

Dabei ist M_{sl} der Morph bezüglich der Ärmellänge (sleeve length), M_{sw} der Morph bezüglich der Ärmelweite (sleeve width), M_{ww} der Morph bezüglich der Taillenweite (waist width) und M_{tl} der Morph bezüglich der Rückenlänge (torso length). Im Prinzip ist es möglich, beliebig viele verschiedene lokale Morphs nach Bedarf zu definieren und diese für den Gesamtmorph entsprechend additiv zusammenzufassen.

Ein Problem, das durch den Morphing-Prozesses auftritt, ist die Entstehung von nicht glatten Übergängen zwischen den einzelnen Segmenten. So entsteht z. B. bei Vergrößerung des Oberarmdurchmessers bei gleichzeitig geringer „wachsendem“ Bekleidungs-Torso eine deutlich sichtbare Treppe im Bereich der Segmenttrennebene. Zur Erreichung von visuell glatten Übergängen werden daher zusätzliche

Korrekturfaktoren in den Morphing-Prozess integriert. Wird beispielsweise die Ärmelweite vergrößert, so müssen die Schulterbreite und der obere Torsobereich entsprechend der Änderung des Ärmels korrigiert werden. Zusätzlich muss der Ärmel noch verschoben werden, um eine Selbstdurchdringung des Kleidungsstückes im Bereich der Achsel zu vermeiden. Für die Änderung des Ärmels lässt sich damit der Gesamtmorph M_{sw} wie folgt definieren:

$$M_{sw} = M_{swu} + M_{swl} + C_{shw} + C_{utw} + T_s$$

Dabei ist M_{swu} der Morph zur Weitenänderung im Bereich des oberen Ärmels, M_{swl} der Morph zur Weitenänderung im Bereich des unteren Ärmels, C_{shw} die Korrektur der Schulterbreite, C_{utw} die Korrektur des oberen Torsobereichs und T_s die Translation des gesamten Ärmels. Der Korrekturmechanismus ist in Abbildung 5-11 schematisch dargestellt.

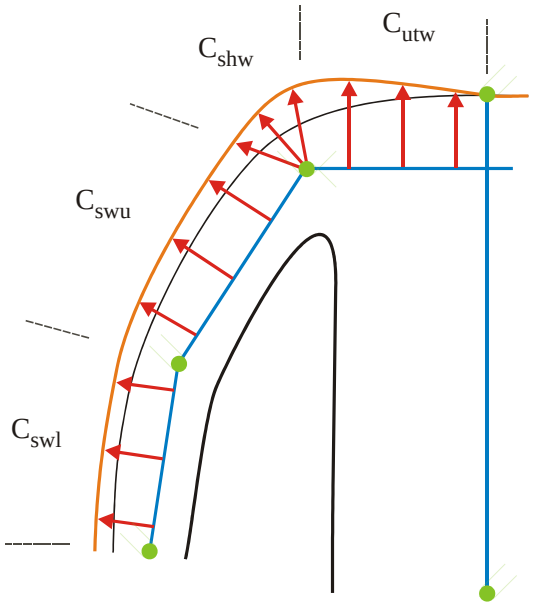

Abbildung 5-11: Integration von Korrekturtermen.

Eine ähnliche Korrektur muss durchgeführt werden, wenn z. B. die Länge des oberen Ärmels geändert wird. In diesem Fall ist dann eine entsprechende Translation des unteren Ärmels notwendig. Für die beschriebene Virtual Try-On Anwendung hat sich herausgestellt, dass das lineare Mischen der Korrekturterme bei benachbarten Teilen für den visuellen Eindruck ausreichend ist.

5.3.4 Erweiterte Morphing-Technologie

Im vorhergehenden Abschnitt wurden die Basisregeln für die Größenänderung von Bekleidungsstücken vorgestellt. Zur Steigerung der realistischen Darstellung muss eine komplexere Behandlung der Maße integriert werden. Außerdem müssen Kohärenzen, die nicht direkt aus den Maßtabellen entnommen werden können, berücksichtigt werden.

Die erste Erweiterung zur komplexeren Behandlung der Maße resultiert aus der folgenden Beobachtung: In der Realität ändert sich der Ärmelumfang nicht stetig entlang des gesamten Ärmels mit demselben Betrag; die Maßzunahme im Oberarm ist beispielsweise stärker als im Unterarm. Dazu definieren in der Bekleidungsindustrie gängige Maßtabellen einzelne Maße für den Ärmelumfang an der Schulter, am Ellbogen und am Handgelenk. Um diese Maße zu interpolieren, muss die oben beschriebene Morphing-Regel für den Morph M_{swu} des oberen Ärmels und den Morph M_{swl} des unteren Ärmels entsprechend erweitert werden. Hierzu wird beim oberen Ärmel zwischen dem Schulter- und dem Ellbogenmaß und beim unteren Ärmel zwischen dem Ellbogen- und dem Handgelenkmaß linear interpoliert. Während z. B. die in Abschnitt 5.3.3 verwendete Morphing-Regel $M_{swu}(x)$ nur von einem Parameter x (konstante Maßänderung über den gesamten Ärmel) abhängig war, ist M_{swu} nun eine Funktion von zwei Variablen: $M_{swu}(y,z)$. Dabei stehen y und z für die Maßänderungen an den jeweiligen oberen und unteren Grenzen des betrachteten Segments. Die Korrektur im Schulterbereich wird weiterhin wie oben beschrieben durchgeführt und ist nun abhängig von der durchgeführten Größenänderung an der oberen Grenze des oberen Ärmels.

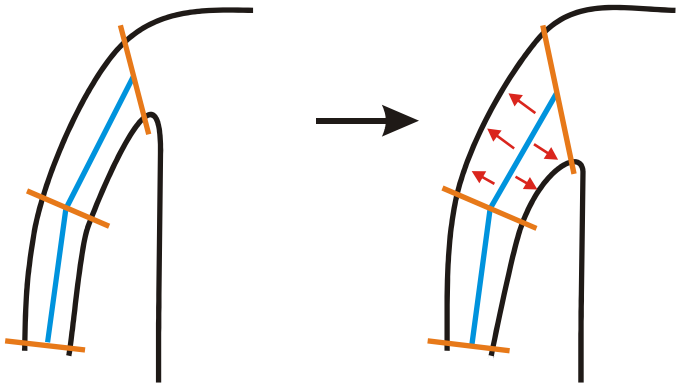

Abbildung 5-12: Interpolation der Maße beim Morph des Ärmels.

Eine weitere Verbesserung betrifft die Berechnung der Weite des Bekleidungs-Torsos und berücksichtigt den realen Produktionsprozess der Oberbekleidung, z. B. eines Jacketts. In der Produktion wird die Weite nicht gleichmäßig um den Körper herum erhöht oder vermindert. Stattdessen bleibt der Rückenteil fast gänz-

lich unverändert und die Weitenänderung wird durch entsprechende Veränderung des linken und rechten Vorderteils erreicht. Diese Information kann also nicht aus den Größentabellen direkt abgeleitet werden, da hier nur die Gesamtweite für eine Größe angegeben wird. Da im Bereich des Torsos eine zylindrische Parametrisierung vorliegt, wird die Morphing-Regel um einen Winkel erweitert. Die Deformation kann damit entsprechend der Winkelposition des betrachteten Torsobereichs jeweils unterschiedlich stark angewandt werden (vergleiche Abbildung 5-13).

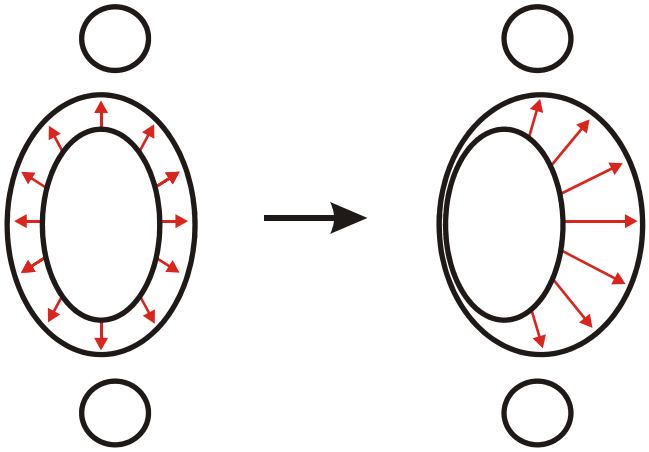

Abbildung 5-13: Verbesserte Berechnung bei Änderung des Torsoumfangs.

Entsprechend der Torsoweite muss auch für die Längenänderung des Torsos die reale Produktion näher betrachtet werden. Auch hier wirkt sich die Änderung nicht auf den gesamten Torso aus, sondern wird nur auf Bereiche angewandt, die unterhalb der Achseln liegen. Begründet ist dies durch die Tatsache, dass eine Längenänderung ansonsten auch unerwünschte Auswirkungen auf den Schulterbereich haben würde, z. B. durch Veränderung der Form der Ärmelausschnitte.

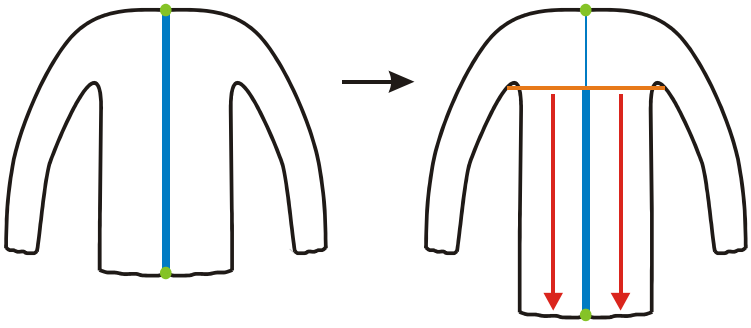

Abbildung 5-14: Verbesserte Berechnung bei Änderung der Rückenlänge.

Wie schon bei der Torsoweite lässt sich diese Eigenart nicht direkt aus den Größentabellen entnehmen. Sie ist aber relativ einfach in die zugehörige Morphing-

Regel integrierbar. Hierzu wird ein Parameterwert *y* eingeführt, welcher die Position einer zum Kleidungsstück waagerechten Ebene in Achselhöhe angibt. Entsprechend wird die Längenänderung nur noch auf die Bereiche unterhalb dieser Ebene angewandt, die Bereiche darüber bleiben unverändert (siehe Abbildung 5-14).

Die nächste Erweiterung betrifft die stärkere Berücksichtigung der Faltenbildung. Wendet man eine Längenänderung, z. B. beim Ärmel, über den kompletten Parameterbereich an, so unterscheidet sich die Faltenbildung je nach Entfernung zum Originalobjekt mehr oder weniger stark. Es ist leicht vorstellbar, dass beispielsweise eine größere Verlängerung des Ärmels zu einer deutlich sichtbaren Glättung der Falten führt, wodurch sich natürlich der Eindruck vom verwendeten Stoff verändert. Dieses Verhalten ist im oberen Teil der Abbildung 5-15 dargestellt. Aus diesem Grund wird die zugehörige Morphing-Regel entsprechend modifiziert: die Längenänderung wird nur noch auf Bereiche angewandt, bei denen keine oder kaum Falten auftreten bzw. das Auftreten einer verstärkten Faltenbildung relativ unwahrscheinlich ist. Am Beispiel des Ärmels tritt bereits bei leichtem Anwickeln des Unterarms in der Regel eine stärkere Faltenbildung im Bereich des Ellbogens auf. Bei der Längenänderung werden daher die Bereiche um Schulter, Ellbogen und Handgelenk unverändert gelassen; der eigentliche Stretch-Vorgang erfolgt somit nur in den dazwischen liegenden Intervallen des Parameterbereichs (siehe Abbildung 5-15 unten). Der visuelle Eindruck des gemorphten Kleidungsstückes kann durch diese Modifikation nochmals gesteigert werden.

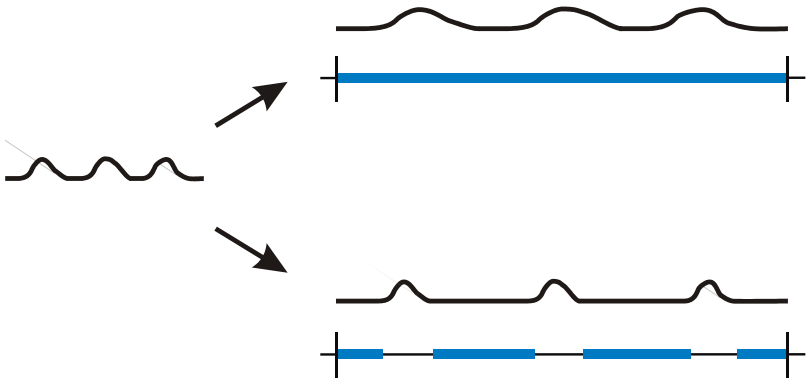

Abbildung 5-15: Modifizierte Parametrisierung zur Erzeugung realistischerer Falten.

Zusammenfassend lässt sich feststellen, dass die Genauigkeit der durch den Morphing-Agenten generierten Kleidungsstücke durch die Integration der beschriebenen Erweiterungsregeln signifikant verbessert werden kann. Entsprechend wird auch der visuelle Eindruck durch die eingeführten Regel-Modifikationen deutlich gesteigert.

5.3.5 Virtuelle Anprobe

Die virtuelle Anprobe eines Kleidungsstückes im Rechner erfolgt in drei aufeinander folgenden Schritten: zunächst muss anhand der einzelnen Körpermaße die benötigte Kleidungsgröße bestimmt werden, anschließend das Grundmodell der Kleidung im Morphing-Prozess auf diese Größe angepasst werden und schließlich die eigentliche virtuelle Anprobe durch eine geeignete Positionierung der Kleidung an der Figurine erfolgen. Für die Zuordnung der korrekten Kleidungsgröße wurde in der vorliegenden Arbeit das Vermessungstool ScanWorX der Firma Human Solutions GmbH [58] eingesetzt. Mit Hilfe dieses Tools ist es möglich, einen 3D-Body-Scan automatisch analysieren zu lassen. Ergebnis dieser Analyse sind verschiedene Einzelmaße (wie Taillenweite, Armlänge, Beinlänge usw.) sowie eine Menge von Featurepunkten (wie Ellbogen, Schulter, Hals usw.; insgesamt können über 50 verschiedene Featurepunkte bestimmt werden). Durch Kombination geeigneter Körpermaße kann der erste Schritt der virtuellen Anprobe durchgeführt, d. h. die benötige Konfektionsgröße für die einzelnen Kleidungsarten berechnet werden. Nachdem das Kleidungsstück wie in den vorangegangenen Abschnitten beschrieben auf die gewünschte Größe gemorpht wurde, kann die eigentliche virtuelle Anprobe erfolgen. Hierzu werden die Featurepunkte der Figurine als Positionierungshilfe herangezogen. Um eine möglichst schnelle Grundpositionierung des betrachteten Kleidungsstückes zu ermöglichen, werden auch für die einzelnen Kleidungsstücke geeignete Featurepunkte, welche zu Featurepunkten der Figurine korrespondieren, definiert. Im Rahmen der Arbeit erfolgte die Definition der Featurepunkte für die Kleidung manuell, dieser Prozess ist jedoch analog der Bestimmung der Körperfeaturepunkte automatisierbar. Die Positionierung der Kleidung sowie notwendige Korrekturen werden im Folgenden am Beispiel eines Jacketts beschrieben, der Prozess lässt sich jedoch in analoger Form auch für alle anderen betrachteten Kleidungsarten verwenden.

In einem ersten Schritt wird das Jackett im Schulterbereich korrekt positioniert. Hierzu werden die entsprechenden Featurepunkte von Körper und Jackett möglichst gut in Übereinstimmung gebracht (siehe Abbildung 5-16, Mitte). Die korrekte Positionierung im Schulterbereich impliziert jedoch nicht zwangsläufig auch eine geeignete Positionierung im Armbereich. Ausgelöst durch unterschiedliche Haltungen während des Scanprozesses ist hier in der Regel eine Situation analog der mittleren Darstellung in Abbildung 5-16 zu erwarten. Die nötige Korrektur kann nun entweder durch Veränderung des Ärmels oder des Arms erfolgen. Da eine Veränderung des Ärmels aufgrund der Sichtbarkeit im Vergleich zum Arm jedoch als deutlich kritischer und zeitaufwendiger zu bewerten ist, wurde hiervon abgesehen.

Abbildung 5-16: Virtuelle Anprobe mit Korrektur der Armpositionen.

Entsprechend wird beim entwickelten Verfahren der (nicht sichtbare) Oberarm komplett ausgeblendet und nur der teilweise sichtbare Unterarm geeignet positioniert. Hierzu wird zunächst die Achse des Unterarms mittels Translation und Rotation in Übereinstimmung mit der Achse des unteren Ärmelteils gebracht und entsprechend der im Bereich des Handgelenkes liegenden Featurepunkte ausgerichtet. Abhängig von der betrachteten Jackettgröße wird durch diesen Prozess jedoch die Armlänge der Figurine beeinflusst, der Arm kann nach der beschriebenen Positionierung also zu kurz oder zu lang sein. Basierend auf der bekannten korrekten Ober- und Unterarmlänge wird daher im Anschluss eine Korrekturoperation angewendet, welche den Unterarm proportional in Achsenrichtung verschiebt. Zusammenfassend wird also der Unterarmstumpf unter Beachtung der korrekten Armlänge im Ärmel des Jacketts positioniert.

Abbildung 5-17: Visuelle Passformkontrolle: unterschiedliche Ärmellängen.

Der beschriebene Prozess bildet hier offensichtlich nicht die Anprobe in der realen Welt ab, da hier die Kleidung immer durch den Körper beeinflusst wird und nicht umgekehrt. Für den Einsatz als 3D-Bekleidungskatalog ist diese Technik jedoch ausreichend, da die zentralen Passformaussagen hiermit visuell getroffen werden

können. Wie die Abbildung 5-17 zeigt, kann beispielsweise die passende Ärmellänge (links: zu kurz, Mitte: korrekte Länge, rechts: zu lang) problemlos beurteilt werden.

Nachdem die Positionierung eines Kleidungsstückes durchgeführt wurde, müssen noch mögliche Durchdringungen mit der Figurine berücksichtigt werden. Der für das Clipping benötigte Zeitaufwand muss jedoch aufgrund der allgemeinen Vorgaben zur Interaktivität so gering wie möglich gehalten werden. Daher werden durch einen Visualisierungsagenten in der Renderingstufe der Visualisierungspipeline diejenigen Körperteile, die komplett durch das Kleidungsstück verdeckt werden, ausgeblendet, indem die zugehörige Geometrie während des Renderingprozesses ermittelt und auf unsichtbar gesetzt wird. Beispielsweise können bei der virtuellen Anprobe eines Sakkos mit Hemd der komplette Oberarm und (fast) der komplette Torso der Figurine ausgeblendet werden. Die Behandlung der Segmente, die teilweise sichtbar sind, ist dagegen komplizierter. In diesem Fall kann ein Standard-Algorithmus zur Erkennung von Durchdringungen eingesetzt werden, gegebenenfalls mit zusätzlichen Optimierungen wie Bounding Volumes oder Octrees. Die hier erkannten Polygone werden dann im Renderprozess ebenfalls auf unsichtbar gesetzt.

Die Behandlung der Kleidungsstücke durch den Clipping-Algorithmus bewirkt aber auch, dass eine Figurine eine viel zu kleine Konfektionsgröße problemlos anprobieren könnte. Da ein solcher Fall in der Realität aber nicht vorkommt, ist beim Design der Benutzeroberfläche darauf zu achten, dass dem Kunden nur im Bereich seiner Größe liegende Bekleidungsgrößen zur Anprobe angeboten werden. Hierdurch kann der Kunde weiterhin ein enger anliegenderes oder locker fallenderes Kleidungsstück virtuell anprobieren, ohne dass ihm durch die Restriktion der Auswahl sinnvolle Kombinationen verloren gehen.

5.4 Umsetzung und Ergebnisse

Die in den vorangegangenen Abschnitten beschriebenen Funktionalitäten wurden in Pure Java unter Beachtung der Java Beans Design Pattern als Komponenten implementiert. Dadurch konnte neben der Verwendung des Visual Prototyping Systems CVP (siehe Kapitel 3.3.3) auch die Steuerung der Visualisierung entsprechend den durch den Client definierten Voraussetzungen (siehe Abschnitt 4.1) in die Virtual Try-On Applikation integriert werden. Die Visualisierung erfolgt komplett in Java 3D, auch hier wurden die benötigten Funktionalitäten komplett als einzelne Java Beans-Bausteine umgesetzt. Neben der 3D-Visualisierung kann das Ergebnis auch im VRML-Format exportiert werden. An dieser Stelle seien nochmals die beiden hiermit in diesem Kontext gewonnenen Hauptvorteile erwähnt: Wiederverwendbarkeit und Flexibilität. Gerade letzteres ist durch den schnellen

Wandel auf dem Grafikmarkt ein äußerst wichtiges Argument, da die Anwendung sehr flexibel auf neue Datenformate oder andere Visualisierungsbibliotheken umgestellt werden kann.

Die prinzipielle Systemstruktur der erstellten Virtual Try-On Applikation (siehe Abbildung 5-18) unterscheidet zwischen den Hauptbausteinen Morphing-Agent, Texturierung und Visualisierung, die jeweils von außen durch verschiedene Events (z. B. zum Laden von Figurine und Kleidung oder zum Auslösen des Morphing-Prozesses) gesteuert werden.

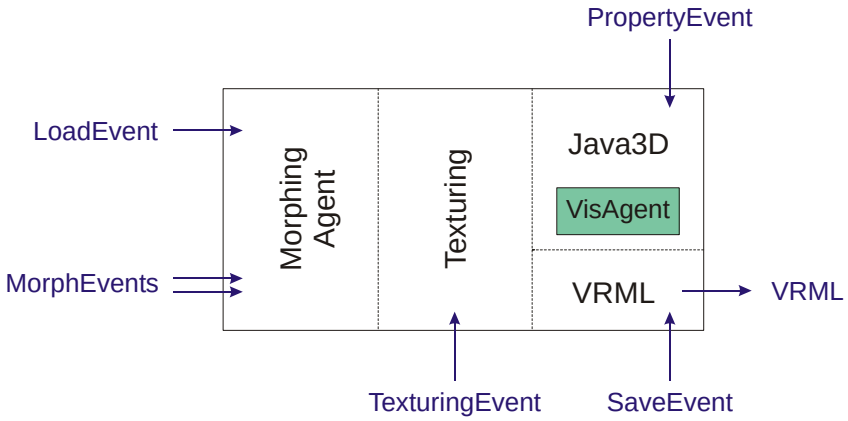

Abbildung 5-18: Prinzipielle Struktur der Virtual Try-On Applikation.

Sämtliche Eingaben, für die das Laden von Datensätzen nötig ist, werden über Load-Events gesteuert:

```
LoadEvent (String filename, String type)
```

Dabei wird durch den Parameter `filename` der zu ladende Datensatz mit absoluter Pfadangabe spezifiziert; Scans können im Human Solutions-internen btr-Format [50] oder im obj-Format [126] geladen werden. Durch den Parameter `type` wird die Art des Datensatzes wie z. B. Hemd, Sakko, Hose oder Größentabelle angegeben, mögliche Typen sind z. B. `scanJacket`, `scanTrousers` und `scanBody`. Ebenfalls über den Load-Event-Mechanismus können Größentabellen im Microsoft Excel csv-Format sowie Featurepunkt-Dateien im Human Solutions-internen vfp-Format [58] importiert werden.

Der eigentliche Morphing-Prozess wird über verschiedene Morph-Events gesteuert, welche die in Abschnitt 5.3.2 beschriebene mehrstufige Entwicklung widerspiegeln. Ein Level0-Morph beschreibt die lokal auf einzelnen Segmenten ausgeführten Morphs, ein Level1-Morph implementiert die notwendigen Korrekturterme, ein Level2-Morph fasst die jeweiligen Level0- und Level1-Morphs unter einem Aufruf zusammen, ein Level3-Morph ermöglicht die Angabe von Einzelän-

derungen in der Maßeinheit mm und ein Level4-Morph erzeugt Konfektionsgrößen basierend auf Größentabellen. Von außen werden in der Regel lediglich der Level4-Morph zur Erzeugung der gewünschten Konfektionsgröße sowie der Level3-Morph zur nachgeschalteten Korrektur der von den Standard-Größentabellen abweichenden Individualmaße durch entsprechende additive Morphs verwendet. Die zugehörigen Events gestalten sich wie folgt:

```
Level4MorphingEvent (String type, Integer newSize)
```

Hier gibt der Parameter `type` den Typ des zu morphenden Bekleidungsstückes (z. B. `Jacket` oder `Trousers`) und der Parameter `newSize` die zu erzeugende Konfektionsgröße an. Die für den Morph nötigen Maße werden dabei vom Morphing-Agenten aus der jeweils geladenen Größentabelle berechnet.

```
Level3MorphingEvent (String identifier, float amount)
```

Der Parameter `identifier` bezeichnet das zu morphende Bekleidungsstücksegment (z. B. `JacketWidthRightArm` oder `TrousersLengthLeftLeg`), während der Parameter `amount` die durchzuführende Einzelmaßänderung in mm angibt (z. B. `4.0` oder `-2.0`).

Das folgende Programmfragment zeigt beispielhaft die nötigen Schritte zum Laden von Bodyscan, Jackett und Hose sowie zur Ausführung der Morphing-Events:

```
// Instanziierung des Morphing-Agenten
MorphingAgent DFKIApp = new MorphingAgent ();

// Laden der benötigten Datensätze
fireLoadEvent ("C:\\Data\\scanb.obj,"scanBody");
fireLoadEvent ("C:\\Data\\features.vfp,"featuresBody");
fireLoadEvent ("C:\\Data\\scan.obj,"scanJacket");
fireLoadEvent ("C:\\Data\\feature_jacket.vfp,
                "featuresJacket");
fireLoadEvent ("C:\\Data\\sizetableJacket.csv,
                "sizetabJacket");
fireLoadEvent ("C:\\Data\\scant.obj,"scanTrousers");
fireLoadEvent ("C:\\Data\\feature_trousers.vfp,
                "featuresTrousers");
fireLoadEvent ("C:\\Data\\sizetableTrousers.csv,
                "sizetabTrousers");

// Ausführung der Morphs
fireMorphingEvent ("Jacket",new Integer(54));
fireAdditionalMorphingEvent ("TrousersLengthLeftLeg",
                12.0);
```

Mit:

```
private void fireLoadEvent (String scan, String type)
{
   LoadEvent e = new LoadEvent (scan,type);
   DFKIApp.HANDLE_LOAD(e);
}

private void fireMorphingEvent (String type,
                                Integer newSize)
{
   Level4MorphingEvent e = new Level4MorphingEvent
                               (type, newSize);
   DFKIApp.HANDLE_LEVEL4_MORPH (e);

}

private void fireAdditionalMorphingEvent
                    (String identifier, float amount)
{
   Level3MorphingEvent e = new Level3MorphingEvent
                              (identifier, amount);
   DFKIApp.HANDLE_LEVEL3_MORPH(e);
}
```

Mittels des obigen Programmfragmentes werden zunächst die Figurine sowie deren Featurepunkte importiert. Anschließend werden Jackett und Hose zusammen mit den zugehörigen Featurepunkten und Größentabellen geladen. Nun wird das Jackett auf die Konfektionsgröße 54 gemorpht sowie das linke Hosenbein anschließend um 12 mm verlängert.

Für die Änderung von bestimmten Eigenschaften des Gesamtmoduls steht zusätzlich ein Property-Event zur Verfügung, der die folgende Syntax besitzt:

```
VTOPropertyEvent (String command)
```

Mit Hilfe der Property-Events können leicht die verschiedenen Eigenschaften der Komponente beeinflusst werden, beispielsweise kann hiermit die Java 3D-Ausgabe zusammen mit den nur in diesem Baustein verwendeten Konvertierungen und Berechnungen abgeschaltet werden, falls nur die VRML-Exportfunktionalität von der steuernden Applikation benötigt wird. Durch den Property-Event-Mechanismus lassen sich zudem leicht neu integrierte optionale Funktionalitäten ansteuern.

Die folgende Abbildung zeigt einen Ausschnitt des interaktiven Individual-Bekleidungskatalogs während der Entstehung im CVP-System. Dabei lässt sich gut der interne Aufbau der Virtual Try-On Applikation mit beispielhafter Steuerung sowie der zugehörige Daten- und Kontrollfluss nachvollziehen. Im linken Bereich sind hier zunächst die Beans zur Erzeugung der einzelnen Menüs bzw. Dialoge, welche zur Steuerung der einzelnen Funktionalitäten benötigt werden, zu

sehen. Die (obere) LoadMenu-Bean gibt dem Benutzer die Auswahlmöglichkeit verschiedener Jacketts und steuert entsprechend der getätigten Auswahl das Laden der Scan-Geometrie (GeometryLoader-Bean) und der zugehörigen Featurepunkte (FeaturePointsLoader-Bean). GeometryLoader- und FeaturePointsLoader-Bean geben dann die initialen Werte für die JacketScan-Bean vor. Durch den mit dieser Bean verknüpften Segmentator wird die geladene Geometrie basierend auf den Featurepunkten für die Anwendung des Morphing-Prozesses segmentiert. Die eigentliche Steuerung des Morphings durch eine geeignete Auswahl und Anwendung der integrierten Morphing-Regeln nimmt dann der Morphing-Agent vor. Dieser erhält seine Ziele vom Benutzer über das Level4MorphingMenu (Vorgabe einer Konfektionsgröße) und / oder das Level3MorphingMenu (Vorgabe von gegebenenfalls additiven Individualmaßen). Als Wissensbasis dient dem Morphing-Agenten die von der SizeTableLoader-Bean gelieferte Größentabelle. Je nach Wahl der entsprechenden Properties kann das Ergebnis dann mittels des Java 3D-Renderers visualisiert oder als VRML- bzw. OBJ-Datei exportiert werden. Die Bean Java3D_Renderer besteht dabei intern aus einer Menge kleiner Bausteine, welche die einzelnen Java 3D-Funktionalitäten zur Verfügung stellen.

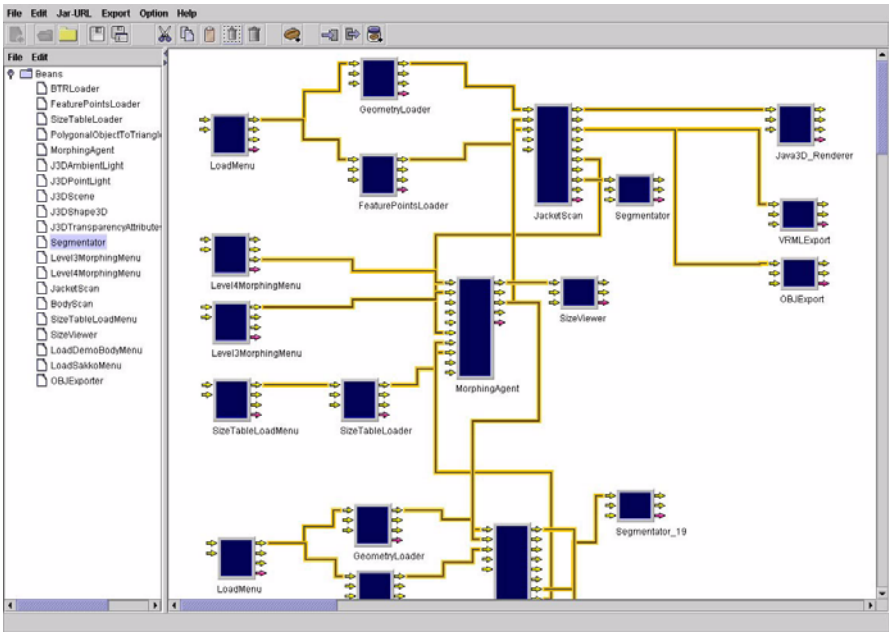

Abbildung 5-19: Virtual Try-On Applikation im CVP-System.

Für Test- und Demonstrationszwecke wurde prototypisch eine virtuelle Shop-Umgebung implementiert (siehe Abbildung 5-20). Das Hauptfenster ist dabei in vier Bereiche unterteilt. Während der Benutzer im linken Bereich die Bekleidungsart und das gewünschte Bekleidungsstück auswählen kann, dient der mittlere Bereich neben der Anzeige der gewählten Kleidungsstücke vor allem zur Aus-

wahl der Größen und Stoffmuster. Der rechte Bereich ist zweigeteilt und enthält die 3D-Visualisierung sowie die Bedienelemente zur Steuerung der virtuellen Anprobe.

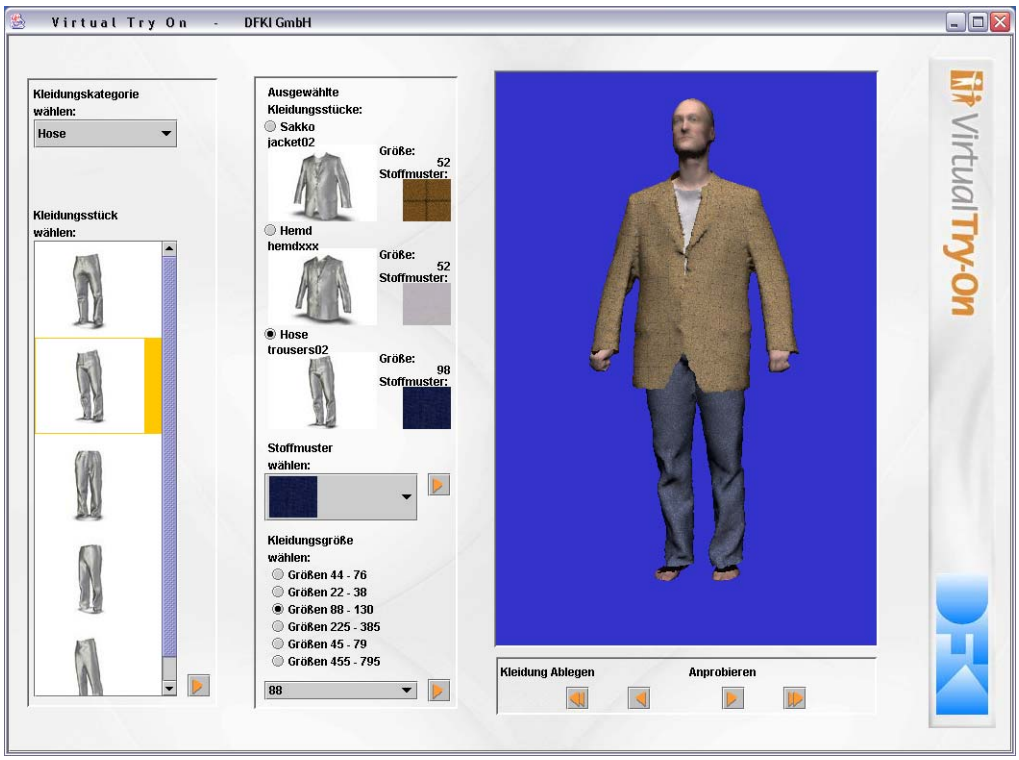

Abbildung 5-20: Prototypische Shop-Umgebung.

Die Abbildungen 5-21 und 5-22 geben einen Eindruck von der mittels der Applikation erzeugten virtuellen Anprobe verschiedener Kleidungsstücke ausgewählter Herren-Oberbekleidung. Dabei trägt der virtuelle Kunde verschiedene Schnitte in verschiedenen Stoffmustern.

Abbildung 5-21: Ergebnisse der virtuellen Anprobe (1).

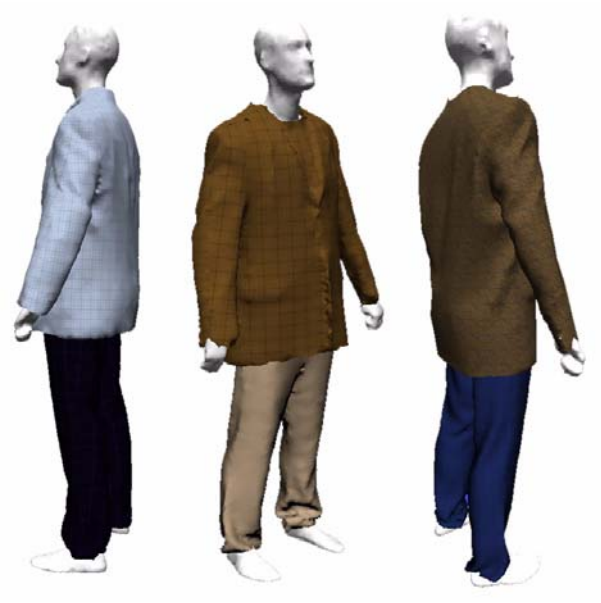

Abbildung 5-22: Ergebnisse der virtuellen Anprobe (2).

6

Zusammenfassung und Ausblick

6.1 Zusammenfassung

Die vorgestellte Übertragung des Begriffes *kontextsensitiv* auf den Bereich der Visualisierung zur Sicherstellung effizienter und kontextbezogener Visualisierungsapplikationen reagiert auf den aktuellen Trend zur besonderen Berücksichtigung von Individualität und Personalisierung in unserer heutigen Informationsgesellschaft.

Durch die innovative Verbindung der beiden Forschungsgebiete Visualisierung und Agententechnologie wurde ein zukunftsorientierter Ansatz für die kontextsensitive Visualisierung geschaffen, der verschiedene Kontexte wie Benutzerkontext, Datenkontext oder Systemkontext als integrale Bestandteile berücksichtigt. Ein besonderes Augenmerk bei der Entwicklung wurde auf die Verwendbarkeit der entstehenden Visualisierungslösungen auf beliebigen, skalierbaren Hardwareplattformen (angefangen von Handys und PDAs bis hin zu Grafik-Workstations) unter optimaler Ausnutzung der jeweils lokal vorhandenen Ressourcen gelegt.

Die praktische Umsetzung basiert auf der Verwendung der Komponententechnologie, die neben der visuellen Erstellung von Visualisierungsanwendungen eine besonders hohe Flexibilität und Wiederverwendbarkeit der entwickelten Softwarebausteine ermöglicht. Neben den Visualisierungsbausteinen sind auch die Agenten als Komponenten umgesetzt und fügen sich somit nahtlos in die Gesamtstruktur ein. Seitens des Benutzers ist kein Unterschied in der Handhabung zwischen den Visualisierungs- und Agentenbausteinen erkennbar, so dass sich die Vorteile der Komponentenorientierung ohne Einschränkungen nutzen lassen.

Die vorgestellten Demonstratoren greifen die einzelnen Kontexte auf. Die Visualisierung großer Datenmengen sowie das vorgestellte skalierbare Informationsvisualisierungssystem für mobile Endgeräte fokussiert dabei vor allem auf System-, Interaktions- und Darstellungskontext, während der interaktive Individual-Bekleidungskatalog die durch den jeweils aktuellen Benutzer- und Datenkontext aufgestellten Vorgaben erfüllt.

6.2 Ausblick

Basierend auf dem Paradigma der kontextsensitiven Visualisierung ist die Konzeption und Entwicklung einer individuell anpassbaren virtuellen Welt geplant, die das visuelle Denken, Abstrahieren und Organisieren von Informationen des Benutzers unterstützt. Das Erreichen dieses Ziels wird durch die synergetische Verschmelzung der Forschungsgebiete kontextsensitive Visualisierung, Information Retrieval und Information Management angestrebt. Das beschriebene Forschungsvorhaben startet an der DFKI GmbH voraussichtlich im Frühjahr 2004.

Anhang A: Abbildungsverzeichnis

Anhang B: Abkürzungsverzeichnis

ACC	Agent Communication Channel
AMS	Agent Management System
API	Application Programming Interface (Anwendungsprogrammierschnittstelle)
AVI	Audio-Video-Interleaved
AWT	Abstract Windowing Toolkit
BNF	Backus-Naur-Form
CBR	Case-based Reasoning
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DARPA	Defense Advanced Research Projects Agency
DCOM	Distributed Component Object Model
DF	Directory Facilitator
DII	Dynamic Invocation Interface
DSI	Dynamic Skeleton Interface
DSL	Digital Subscriber Line
ECOOP	European Conference on Object-Oriented Programming
FIPA	Foundation for Intelligent Physical Agents
GUI	Graphical User Interface
GUID	Globally Unique Identifier
HP	Hewlett Packard
IBM	International Business Machines Corp.
ID	Identification
IDL	Interface Identification Language
IID	Interface Identifier

IPMT	Internal Platform Message Transport
JAR	Java Archive File Format
JDK	Java Developement Kit
JVM	Java Virtual Machine
KI	Künstliche Intelligenz
KIF	Knowledge Interchange Format
KQML	Knowledge Query and Manipulation Language
KRSS	Knowlegde Representations Systems Standards
KSE	Knowledge Sharing Effort
LOD	Level of Detail
MAS	Multi-Agenten-System
MPEG	Motion Picture Expert Group
MSDN	Microsoft Developer Network
OLE	Object Linking and Embedding
OMA	Object Management Architecture
OMG	Object Management Group
ORB	Object Request Broker
PC	Personal Computer
RMI	Remote Method Invocation
URL	Uniform Resource Locator
VKI	Verteilte Künstliche Intelligenz
VTK	Visualization Toolkit
VTable	Virtual Function Table
WAV	Waveform sound files
WWW	World Wide Web

Anhang C:Lebenslauf

Zur Person

Name:	Achim Ebert
geboren am:	20. September 1968 in Ingelheim am Rhein
Familienstand:	ledig
Nationalität:	Deutscher

Schulbildung

1975 - 1979	Pestalozzi Grundschule, Ingelheim
1979 - 1988	Sebastian Münster Gymnasium, Ingelheim (Leistungskurse: Mathematik, Physik, Englisch)

Wehrdienst

1988 - 1989	Grundwehrdienst in Bergisch-Gladbach und Lahnstein

Hochschulstudium

1989 - 1996 Studium der Informatik mit Nebenfach Mathematik an der Universität Kaiserslautern

Diplomprüfungen:

- Computergrafik und CAGD
- Scientific Visualization und Geometric Modeling
- Reduktionssysteme
- Rechnerstrukturen

Diplomarbeit:

Eine objektorientierte Basisbibliothek zur Generierung rationaler Kurven und Flächen

Praktische Tätigkeiten

1992 - 1996	Wissenschaftliche Hilfskraft (Vorlesungs-, Übungs-, Praktika- und Proseminarbetreuung, projektbezogene Programmiertätigkeiten)
1996 - 1999	Wissenschaftlicher Mitarbeiter an der Universität Kaiserslautern im Fachbereich Informatik, Arbeitsgruppe "Grafische Datenverarbeitung und Computergeometrie" in den Projekten Viper, Vicom und Anthropoid, zudem verantwortlich für Administration und Web-Präsenz
seit 1999	Wissenschaftlicher Mitarbeiter am Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI) GmbH, Kaiserslautern im Forschungsbereich "Intelligente Visualisierung und Simulation", Leitung der Abteilung "Immersive VR"

Forschungsinteressen

Intelligente Visualisierungssysteme
Komponentenbasierte Visualisierung
Agentensysteme
Kleidungsmodellierung
Virtual Reality Systeme
Mobile Visualisierung

Publikationen

Barthel, H., Bender, M., Divivier, A., Ebert, A., Hagen, H.
A Component based Visualization Architecture
CoData, Baveno, 2000

Bender, M., Ebert, A., Rodrian, H.-C., Klein, R.
A Hybrid Web-based Toolkit for Human Modeling
Dagstuhl ’97, Scientific Visualization Proceedings, 1997

Bender, M., Klein, R., Disch, A., Ebert, A.
A Functional Framework for Web-based Information Visualization Systems
IEEE Transactions on Visualization and Computer Graphics, Vol. 6, No. 1, January/March 2000

Divivier, A., Ebert, A., Barthel, H., Hagen, H., Bender, B.
Belt Design Service – A Human Factors Application
IASTED International Conference on Visualization, Imaging and Image Processing (VIIP 2002), Malaga, Spain, 2002

Divivier, A., Ebert, A., Bender, M., Hagen, H.:
Seat Belt Routings over Virtual Humans – The Belt Design Service
IEEE Visualization 2002, Boston, Massachusetts, USA, 2002

Ebert, A., Divivier, A., Bender, M., Barthel, H.
A Visualization System using Multi Agent Technologies
IEEE Visualization Conference, Salt Lake City, 2000

Ebert, A., Bender, M., Barthel, H., Divivier, A.
Tuning a Component-based Visualization Architecture by Agents
1st International Symposium on Smart Graphics, New York, 2001

Ebert, A., Divivier, A., Barthel, B., Bender, M., Hagen, H.
Improving Development and Usage of Visualization Applications
IASTED International Conference on Visualization, Imaging and Image Processing (VIIP 2001), Marbella, 2001

Ebert, A., Ginkel, I., Barthel, H., Divivier, A., Bender, B.
Efficient Assistance Of Virtual Dress Fitting Using Intelligent Morphing
IASTED International Conference on Visualization, Imaging and Image Processing (VIIP 2002), Malaga, Spain, 2002

Ebert, A., Ginkel, I., Hagen, H.
Rule-based Morphing Techniques for Interactive Clothing Catalogs
Dagstuhl 2003, Scientific Visualization Proceedings, 2003

Ebert, A., Schädlich, J., Disch, A.
Innovative Retexturing Using Cooperative Patterns
IASTED International Conference on Visualization, Imaging and Image Processing (VIIP 2003), Malaga, Spain, 2003

Hagen, H., Barthel, H., Ebert, A., Divivier, A., Bender, M.
A Component- and Multi Agent-based Visualization System Architecture
IMC Rostock-Warnemünde, 2000

Hagen, H., Barthel, H., Ebert, A., Divivier, A., Bender, M.
Component-based Intelligent Visualization
Dagstuhl 2000, Scientific Visualization Proceedings

Hagen, H., Divivier, A., Barthel, H., Ebert, A., Bender, M.
MacVis – A System Architecture for Intelligent Component-based Visualization
Workshop on New Paradigms in Information Visualization and Manipulation, Washington D.C., 2000

Hagen, H., Ebert, A., van Lengen, R. H., Scheuermann, G..
Scientific Visualization – Methods and Applications
Informatics 2001, 2001

Reuther, B., Ebert, A., Müller, P., Hagen, H.
A Powerful Communication Framework for Distributed Virtual Reality
IEEE Visualization 2002, Boston, Massachusetts, USA, 2002

Schuchardt, L., Steinmetz, H., Ehret, J., Ebert, A., Schmitt, T. G.
Scalable mobile information system to support the treatment process and the workflow of wastewater facilities.
Eingereicht zur International IWA Conference on Automation in Water Quality Monitoring, 2004

Steinmetz, H., Schuchardt, L., Wiese, J., Ehret, J., Ebert, A., Schmitt, T.G.
Einsatz mobiler Endgeräte zur effizienten Informationsvermittlung im Bereich von Abwasseranlagen
Gemeinschaftstagung "Mess- und Regelungstechnik in abwassertechnischen Anlagen", ATV-DVWK, 2003

Anhang D:Literaturverzeichnis

[1] Advanced Visualization Systems Inc.: *Active X Visualization Components*. Technology White Paper, 1999.

[2] Advanced Visualization Systems Inc.: *AVS/Express: Gaining Insight Through Data Visualization*. Technology White Paper, 1999.

[3] Advanced Visualization Systems Inc.: *AVS Technical Overview.*

[4] Advanced Visualization Systems Inc.: *AVS Users Guide*.

[5] Barthel, H.: *Komponentenbasiertes visuelles Prototyping von Visualisierungsanwendungen*. Dissertation, Fachbereich Informatik, Universität Kaiserslautern, 2000.

[6] Barthel, H., Bender, M., Divivier, A., Ebert, A., Hagen, H.: *A Component based Visualization Architecture*. CoData, Baveno, 2000.

[7] Bender, M.: *A Functional Framework for Efficient Web-based Scientific Visualization Systems*. Dissertation, Fachbereich Informatik, Universität Kaiserslautern, 2000.

[8] Bender, M., Brill, M.: *Computer-Grafik – Ein anwendungsorientiertes Lehrbuch*. Carl Hanser Verlag, 2003.

[9] Bender, M., Ebert, A., Rodrian, H.-C., Klein, R.: *A Hybrid Web-based Toolkit for Human Modeling*. Dagstuhl ’97, Scientific Visualization Proceedings.

[10] Bethel, E.W., Uselton, S.: *Shape distortion in computer-assisted keyframe animation*. Computer Animation 89, 1989.

[11] Breen, D. E., House, D. H., Wozny, M. J.: *Predicting the Drape of Woven Cloth Using Interacting Particles*. European Computer Industry Research Centre, ECRC 94(16), 1994.

[12] Breen, D. E., House, D. H., Wozny, M. J.: *A Particle Based Model for Simulating the Draping Behavior of Woven Cloth*. European Computer Industry Research Centre, ECRC 94(19), 1994.

[13] Brustoloni, J. C.: *Autonomous Agents – Characterization and Requirements*. Technical Report CMU-CS-91-204, Carnegie Mellone University, Pittsburgh.

[14] Capurro, R.: *Einführung in den Informationsbegriff*. 1999.

[15] Chavez, A., Maes, P.: *Kasbah: An Agent Marketplace for Buying and Selling Goods*. Proceedings of the First International Conference on the Practi-

cal Application of Intelligent Agents and Multi-Agent Technology, London, 1996.

[16] Cockburn, D., Jennings, N. R.: *ARCHON: A Distributed Artificial Intelligence System for Industrial Applications*. Foundations of Distributed Artificial Intelligence, Wiley, 1996.

[17] The DARPA Knowledge Sharing Initiative External Interfaces Working Group: *Specification of the KQML Agent-Communication Language*. Draft, 1993.

[18] Denzinger, J.: *Grundlagen von Multi-Agenten-Systemen*. Universität Kaiserslautern, Kaiserslautern.

[19] Denzinger, J., Fuchs, M.: *Experiments in Learning Prototypical Situations for Variants of the Pursuit Game*. ICMAS Proceedings, Kyoto, 1996.

[20] *Duden – Die Deutsche Rechtschreibung*. Duden Band 1, 22. Auflage, Bibliographisches Institut & F. A. Brockhaus AG, 2000.

[21] Dunlop, R. L.: *DirectX 7 Programmierung*. Markt+Technik Verlag, 2000.

[22] Eberhardt, B., Weber, A., Strasser, W.: *A Fast, Flexible, Particle System Model for Cloth Draping*. IEEE Computer Graphics and Applications, 1996.

[23] Ebert, A., Divivier, A., Bender, M., Barthel, H.: *A Visualization System using Multi Agent Technologies*. IEEE Visualization Conference, Salt Lake City, 2000.

[24] Ebert, A., Bender, M., Barthel, H., Divivier, A.: *Tuning a Component-based Visualization Architecture by Agents*. 1st International Symposium on Smart Graphics, New York, 2001.

[25] Ebert, A., Ginkel, I., Barthel, H., Divivier, A., Bender, M.: *Efficient Assistance Of Virtual Dress Fitting Using Intelligent Morphing*. IASTED International Conference on Visualization, Imaging and Image Processing (VIIP 2002), Malaga, Spain, 2002.

[26] Ebert, A., Ginkel, I., Hagen, H.: *Rule-based Morphing Techniques for Interactive Clothing Catalogs*. Dagstuhl 2003, Scientific Visualization Proceedings, 2003.

[27] Ebert, A., Schädlich, J., Disch, A.: *Innovative Retexturing Using Cooperative Patterns*. IASTED International Conference on Visualization, Imaging and Image Processing (VIIP 2003), Malaga, Spain, 2003.

[28] Eck, M., De Rose, T., Duchamp, T., Hoppe, H., Lounsbery, M., Stuetzle, W.: *Multiresolution analysis of arbitrary meshes*. ACM SIGGRAPH'95, 1995.

[29] epro GmbH: *inVISU PMS – Im Überblick: Einfach projektieren statt programmieren mit inVISU PMS*. Version 1.1.

[30] epro GmbH: *inVISU PMS – Technische Daten*. Version 3.1.

[31] Fellner, W. D.: *Computergraphik*. BI Wissenschaftsverlag, Reihe Informatik, Band 58, 2. Auflage, 1992.

[32] *Foundation for Intelligent Physical Agents (FIPA)Homepage*. http://www.fipa.org/.

[33] Foundation for Intelligent Physical Agents: *Agent Management*. FIPA 98 Specification, Version 1.0, Part 1, 1998.

[34] Foundation for Intelligent Physical Agents: *FIPA Agent Management Specification*. Document number XC00023G (Experimental), 2000.

[35] Foley, J. D., van Dam, A., Feiner, S. K., Hughes, J. F.: *Computer Graphics – Principles and Practice*. Addison-Wesley, 1990.

[36] Foner, L.: *What's an Agent, Anyway? A Social Case Study.* Agents Memo 93-01.

[37] Franklin, S. P., Graesser, A.: *Is it an Agent, or just a Program? – A Taxonomy for Autonomous Agents*. Proceedings of the Third International Workshop On Agent Theories, Architectures, and Languages, Springer Verlag, 1996.

[38] Futuremark Corporation: *3D Mark*. http://www.futuremark.com/.

[39] Genesereth, M. R., Fikes, R. E.: *Knowledge Interchange Format Version 3.0 Reference Manual*. Stanford University, California, Logic Group, Report Logic-92-1, June 1992.

[40] Genesereth, M. R., Ketchpel, S. P.: *Software Agents*. Stanford University, California, Logic Group.

[41] Ginkel. I.: *Physikalische Methoden zur Modellierung und Visualisierung von Stoffen und Kleidung*. Projektarbeit, Fachbereich Informatik, Universität Kaiserslautern, 1999.

[42] Ginkel. I.: *Computergraphische Methoden zur Modellierung und Visualisierung von Stoffen und Kleidung*. Seminar, Fachbereich Informatik, Universität Kaiserslautern, 1998.

[43] Griffel, F.: Componentware: *Konzepte und Techniken eines Softwareparadigmas*. dpunkt-Verlag, 1998.

[44] Haber, R. D., McNabb, D. A.: *Visualization Idioms: A Conceptional Model for Scientific Visualization*. Visualization in Scientific Computing, IEEE Press, 1990.

[45] Hagen, H., Barthel, H., Ebert, A., Divivier, A., Bender, M.: *A Component- and Multi Agent-based Visualization System Architecture*. IMC Rostock-Warnemünde, 2000.

[46] Hagen, H., Barthel, H., Ebert, A., Divivier, A., Bender, M.: *Component-based Intelligent Visualization*. Dagstuhl 2000, Scientific Visualization Proceedings.

[47] Hagen, H., Divivier, A., Barthel, H., Ebert, A., Bender, M.: *MacVis – A System Architecture for Intelligent Component-based Visualization*. Workshop on New Paradigms in Information Visualization and Manipulation, Washington D.C., 2000.

[48] Hagen, H., Ebert, A., van Lengen, R. H., Scheuermann, G..: *Scientific Visualization – Methods and Applications*. Informatics 2001, 2001.

[49] Hagen, H., Müller, H., Nielson, G. M.: *Scientific Visualization – Overviews, Methodologies, Techniques*. IEEE, 1997.

[50] Hansen, G.., Hamfeld, H.: *Binary Scan Format 1.0*. Tecmath AG, 1998.

[51] Harmon, P.: *Components Development Strategies*. Cutter Information Corp. on Managing and Developing Component-Based Systems, 1998.

[52] Hayes-Roth, B., Washington, R., Ash, D., Hewett, R., Collinot, A., Vina, A., Seiver, A.: *Guardian: A prototype intelligent agent for intensive-care monitoring*. Journal of Artificial Intelligence in Medicine, 4, 1992.

[53] Heuer-Hasenpatt, H., Hollunder, B., Kittlaus, H.-B., Schumacher, N.: *Bausteinorientierte Anwendungsentwicklung*. SIZ GmbH, 1998.

[54] Hinds, B. K., McCartney, J.: *Interactive garment design*. The Visual Computer, 1990.

[55] Hoppe, H., De Rose, T., Duchamp, T., Mc Donald, J., Stuetzle, W.: *Mesh optimization*. ACM SIGGRAPH'93, 1993.

[56] Huang, J., Jennings, N. R., Fox, J.: *An Agent Architecture for Distributed Medical Care*. Intelligent Agents, Lecture Notes in Artificial Intelligence, Springer Verlag, 1995.

[57] Huges, J.: *Scheduled fourier volume morphing*. Computer Graphics, 1992.

[58] Human Solutions GmbH: *ScanWorX*. http://www.human-solutions.de/.

[59] IBM Research: *IBM Visualization Data Explorer Documentation*. Version 3 Release 1 Modification 4.

[60] IBM Research: *Open Visualization Data Explorer Documentation: Building Applications*.

[61] Janson, S.: *Intelligent Software Agents*. http://www.agentbase.com/survey.html.

[62] Jennings, N. R.: *The ARCHON System and its Applications*. Second International Working Conference on Cooperating Knowledge Based Systems Keele, 1994.

[63] Jennings, N. R., Corera, J. M., Laresgoiti, I.: *Developing Industrial Multi-Agent Systems*. Proceedings of the First International Conference on Multi-agent Systems, 1995.

[64] Jennings, N. R., Wooldridge, M.: *Agent Technology: Foundations, Applications, and Markets*. Springer Verlag, 1998.

[65] Kant, I.: *Beantwortung der Frage: Was ist Aufklärung?*. Berlinische Monatsschrift, Dezember-Heft, 1784.

[66] Kaul, A., Rossignac, J.: *Solid-interpolating deformation: Construction and animation of pips*. Eurographics 91, 1991.

[67] Kent, J., Carlson, W., Parent, R.: *Shape Transformations for Polyhedral Objects*. ACM Computer Graphics, Vol. 26, No. 2, 1992.

[68] Khoral Inc.: *Cantata: The Visual Programming Environment for the Khoros System*. White Paper.

[69] Khoral Inc.: *Khoros: An Integrated Development Environment for Scientific Computing & Visualization*. White Paper.

[70] King, J. A.: *Intelligent Agents – Bringing Good Things To Life*. AI-Expert, Februar 1995.

[71] Kitware Inc.: *VTK Homepage*. http://www.kitware.com/vtk.html.

[72] Kronenberger, M.: *Quadric-basierte Kantenkontraktion*. Projektarbeit, Fachbereich Informatik, Universität Kaiserslautern, 2001.

[73] Larsson, J. E., Hayes-Roth, B., Gaba, D.: *Guardian: Final evaluation*. Technical Report KSL-96-25, Knowledge Systems Laboratory, Stanford University, 1996.

[74] Lazarus, F., Verroust, A.: *Feature-based shape transformation for polyedral objects*. Research Report INRIA, 1994.

[75] Ljungberg, M., Lucas, A.: *The OASIS air traffic management system*. Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence, 1992.

[76] Lorensen, B., Cline, H.: *Marching Cubes: A High Resolution 3D Surface Construction Algorithm*. ACM SIGGRAPH '87, 1987.

[77] Maes, P.: *Agents that Reduce Work and Information Overload*. Communications of the ACM, 37(7), 1994.

[78] Maes, P., Guttman, R., Moukas, A.: *Agents that Buy and Sell: Transforming Commerce as We Know It*. Communications of the ACM 42, 3, 1999.

[79] *Mesa 3D Graphics Library*. http://www.mesa3d.org/.

[80] *Meyers Grosses Taschenlexikon in 24 Bänden*. Bibliographisches Institut Mannheim/Wien/Zürich, Meyers Lexikonverlag, 1983.

[81] Microsoft Corporation: *DirectX Homepage*. http://www.microsoft.com/directx/.

[82] Microsoft Corporation: *MSDN Library – Graphics and Multimedia: DirectX 9.0 SDK*.

[83] Microsoft Corporation: *MSDN Library – Visual Studio 6.0*.

[84] Nielson, G., Hamann, B.: *The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes*. Visualization '91, 1991.

[85] The Numerical Algorithms Group (NAG) Ltd.: *IRIS Explorer Module Writer's Guide*, Release 5.0 Edition.

[86] The Numerical Algorithms Group (NAG) Ltd.: *IRIS Explorer Tutorial*, Release 5.0 Edition.

[87] The Numerical Algorithms Group (NAG) Ltd.: *IRIS Explorer User's Guide*, Release 5.0 Edition.

[88] Nwana, H. S.: *Software Agents – An Overview*. The Knowledge Engineering Review, 11(3), 1996.

[89] *Object Management Group (OMG) Homepage*. http://www.omg.org/.

[90] *OpenGL Homepage*. http://www.opengl.org/.

[91] Parent, R.: *Shape transformation by boundary representation interpolation: a recursive approach to establishing face correspondences*. The Journal of Visualization and Computer Animation, 1992.

[92] Parunak, H. V. D.: *Manufactoring experience with the contract net*. Distributed AI, 1987.

[93] Piemont, C.: *Komponenten in Java: Einsatz und Entwicklung von Java Beans mit VisualAge For Java*. d-punkt-Verlag, 1999.

[94] Ranze, K. C., Müller, H. J.: *Über den Einsatz von Agenten in Umweltanwendungen*. Technologie-Zentrum Informatik, Universität Bremen.

[95] Rechenberg, P., Pomberger, G.. (Hrsg.): *Informatik-Handbuch*. Carl Hanser Verlag München Wien, 1999.

[96] San Diego Supercomputer Center: *The Java 3D Repository.* http://java3d.sdsc.edu/.

[97] Schädlich, J.: *Triangulierung und Texturierung von 3D-Scandaten*. Projektarbeit, Fachbereich Informatik, Universität Kaiserslautern, 2000.

[98] Schroeder, W. J., Martin, K. M., Avila, L. S., Law, C. C.: *Vtk User's Guide: May 2000*. Kitware, Inc., 2000.

[99] Schroeder, W., Martin, K., Lorensen, W. E.: *The Design and Implementation of an Object-Oriented Toolkit for 3D Graphics and Visualization*. IEEE Visualization Conference, San Francisco, 1996.

[100] Schroeder, W. J., Martin, K. M., Lorensen, B.: *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics, 2nd Edition*. Prentice Hall, 1998.

[101] Schuchardt, L., Steinmetz, H., Ehret, J., Ebert, A., Schmitt, T. G.: *Scalable mobile information system to support the treatment process and the workflow of wastewater facilities*. Angenommen bei der International IWA Conference on Automation in Water Quality Monitoring, 2004.

[102] Schütze, M., Alex, J., Jumar, U., Ogurek, M.: *Prozessdatenvisualisierung über mobile Endgeräte in der Ver- und Entsorgungstechnik*. Proceedings der Tagung "Simulation und Visualisierung 2003", Magdeburg, 2003.

[103] Segal, M., Akeley, K.: *The Design of the OpenGL Graphics Interface*. Silicon Graphics Computer Systems, 1994.

[104] Segal, M., Akeley, K.: *The OpenGL graphics system: A specification*. Technical report, Silicon Graphics Computer Systems, 1992.

[105] Shreiner, D. (Edt.): *OpenGL Reference Manual: The Official Reference Document to OpenGL, Version 1.2*. Addison-Wesley, 1999.

[106] Sirotin, V., Debeloff, V., Urri, Y.: *DirectX-Programmierung mit Visual C++*. Addison-Wesley, 1999.

[107] Smith, R.: *The contract net protocol: High-level communication and control in distributed problem solver.* IEEE Transactions on Computers, 29(12), December 1980.

[108] Sowizral, H., Rushforth, K., Deering, M.: *The Java 3D API Specification*. Addison-Wesley 1998.

[109] Steele, G.: *Common Lisp: The Language*. 2nd Edition, 1990.

[110] Steinmetz, H., Schuchardt, L., Wiese, J., Ehret, J., Ebert, A., Schmitt, T. G.: *Einsatz mobiler Endger‰te zur effizienten Informationsvermittlung im Bereich von Abwasseranlagen*. Gemeinschaftstagung "Mess- und Regelungstechnik in abwassertechnischen Anlagen", ATV-DVWK, 2003.

[111] Steinmetz, R.: *Multimedia-Technologie, Einführung und Grundlagen*. Springer Verlag Berlin Heidelberg, 1993.

[112] Sun Microsystems Inc.: *Java Beans*. Spezifikation, 1997.

[113] Sun Microsystems Inc.: *Java Beans for Java Studio: Architecture and API*. 1997.

[114] Sun Microsystems Inc.: *Java Homepage*. http://java.sun.com/.

[115] Sun Microsystems Inc.: *Java 3D API Collateral – The Java 3D API Specification Version 0.98*.

[116] Sun Microsystems Inc.: *Java 3D API Documentation 1.3.1*.

[117] Taillefer, F.: *Mixed Modeling*. Proceedings of Compugraphics (First International Conference on Computational Graphics and Visualization Techniques), 1991.

[118] Terzopoulos, D., et al.: *Elastically Deformable Models*. Computer Giaphics, Volume 21, Number 4, 1987.

[119] UMBC (University of Maryland Baltimore County) Laboratory for Advanced Information Technology: *KQML Homepage*. http://www.cs.umbc.edu/kqml/.

[120] Upson, C., et al.: *The Application Visualisation System: A Computational Environment for Scientific Visualization*. IEEE Computer Graphics and Applications, 1989.

[121] Verbundprojekt *Virtual Try-On*. http://www.virtualtryon.de/.

[122] Vitronic: *VITUS – 3D-Ganzkörperscanner.* http://www.vitus.de/.

[123] Vitronic: *Vitus Bodyscanner – kompakt und smart*.

[124] Volino, P., Thalmann, N., Jianhua, S., Thalmann, D.: *An Evolving System for Simulation Clothes an Virtual Actors*. IEEE Computer Graphics and Applications, 1996.

[125] Walton, J.: *Data Visualization with IRIS Explorer – What's New?*. NAG Ltd., Oxford, UK, 1996.

[126] Wavefront: *Object files*. OBJ-Format-Spezifikation.

[127] Wavish, P., Graham, M.: *A situated action approach to implementing characters in computer games*. Applied Artificial Intelligence, 10(1), 1996.

[128] Weil, J.: *The Synthesis of Cloth Objects*. Computer Graphics (Proc. ACM SIGGRAPH), Volume 20, 1986.

[129] Woo, M., Neider, J., Davis, T., Shreiner, D.: *OpenGL 1.2 Programming Guide, Third Edition: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley, 1999.

[130] Wooldridge, M., Jennings, N. R.: *Intelligent Agents: Theory and Practice*. Knowledge Engineering Review, 10(2), 1995.

[131] Wright Jr., R. S., Sweet, M.: *OpenGL SuperBible, Second Edition*. Waite Group Press, 1999.