Interaktive Simulation und Visualisierung in der computergestützten Biochemie

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachbereich Informatik und Mathematik
der Johann-Wolfgang-Goethe - Universität
in Frankfurt am Main

von
Christian Seiler
aus Groß-Gerau

Frankfurt 2006
DF1

vom Fachbereich Informatik und Mathematik der
Johann Wolfgang Goethe – Universität als Dissertation angenommen.

Dekan:	Prof. Dr.-Ing. Detlef Krömker
Gutachter:	Prof. Dr.-Ing. Detlef Krömker, Prof. Dr. Gisbert Schneider
Datum der Disputation:	2. März 2006

Kurzfassung

Die moderne Biochemie ist eine Wissenschaft, die sich im Wandel befindet. Während die bisherige Forschung sehr stark experimentell geprägt ist, existiert eine theoretische Biologie, analog zur theoretischen Chemie, nur in Ansätzen. Trotzdem wandelt sich auch diese Wissenschaft hin zu einer stärkeren Einbindung theoretischer Ansätze. Der Grund hierfür liegt in der Betrachtung von zunehmend komplexeren Systemen. So beschäftigt man sich in der Systembiologie, einem Teilbereich der Biochemie, unter anderem mit der Aufklärung komplexer Reaktionsnetzwerke. Während Ausschnitte dieser Netzwerke weiterhin experimentell aufgeklärt und verstanden werden, lässt sich das zusammenhängende Bild zunehmend nur noch durch eine theoretisch geprägte Modellbildung fassen. Darüber hinaus zeigen neuere Forschungsergebnisse [112] die Bedeutung der Tatsache, dass Moleküle, Zellen und Zellhaufen, also wichtige Forschungsubjekte der Biochmie, dreidimensionale Gebilde sind – eine Tatsache, die bei der Modellbildung berücksichtigt werden muss. Eine Antwort auf die genannten Herausforderungen ist der konzertierte Einsatz von *Simulation* und *Visualisierung* als Mittel des Erkenntnisgewinns. Damit ist die Informatik gefordert entsprechende dedizierte Werkzeuge zu entwickeln, die Simulation, Visualisierung und Interaktion im Kontext des von der Anwendungsdisziplin gesetzten räumlich-zeitlichen Problemkreises miteinander verbinden.

In dieser Arbeit wird ein integriertes Konzept zu Simulation, Interaktivität und Visualisierung vorgelegt, das auf einer Anforderungsanalyse in Bezug auf *Anforderungen an die Simulation* und *Anforderungen an die Interaktivität und Visualisierung* basiert. Zur Lösung der aufgeworfenen Probleme wird ein „Baukastensystem“ auf Basis von Multi-Agenten-Systemen vorgeschlagen. Die Auswahl des geeigneten Simulationsverfahrens, z. B. die Auswahl eines stochastischen Verfahrens gegenüber einem deterministischen Verfahrens, wird so zur Auswahl eines Bausteins, wobei gezeigt wird, wie z. B. mit Hilfe von Regeln die Auswahl auch automatisiert werden kann. Ebenso wird gezeigt, wie man „Baussteine“ auch im räumlichen Sinne verstehen kann, als Dinge, die in einem dreidimensionalen Kontext einen bestimmten Raum einnehmen und die, in ihrer Gesamtheit betrachtet, den Beobachtungsraum der Simulation ausfüllen. Diese Bausteine finden sich entsprechend ebenfalls im Kontext der Interaktion wieder. Ein wichtiger Aspekt in diesem Baukastenkonzept ist die Frage der Kommunikationsstruktur und des Kommunikationsprotokolls, für den ein Vorschlag erarbeitet wird. Das entwickelte Gesamtkonzept besteht aus zwei Teilen: Einem Konzept für Ein- und Ausgabegeräte mit einer gemeinsamen Metapher, die die Geräte logisch in den Anwendungskontext einbettet und einem Simulations- und Visualisierungskonzept auf der Basis der Kopplung heterogener intelligenter Agenten in eine gemeinsame Simulationsumgebung. Hierfür wurde ein spezieller Dialekt einer Agentenkommunikationssprache entwickelt, der dabei insbesondere den Aspekt der dreidimensionalen Visualierung einer solchen Simulation berücksichtigt.

Die vorgestellten Konzepte wurden in einer Reihe von Anwendungen umgesetzt. Das Interaktionskonzept wurde etwa durch ein neues Ein-/ Ausgabegerät, die *Virtual Glove Box*, realisiert. Darüber hinaus wird das Gesamtkonzept durch eine Reihe softwaretechnischer Umsetzungen belegt. So wurden neben der Softwareseite der Virtual Glove Box in Form einer graphischen Benutzungsoberfläche mit haptischer Unterstützung, eine komponentenbasierte Molecular-Modeling-Anwendung, ein 3D Diffusions-Reaktionssystem, sowie eine Editorandwendung für Zelldifferenzierungssimulationen entwickelt.

Die Validierung der Realisierungen anhand der in der Anforderungsanalyse gewonnenen Erkenntnisse, ergibt die Tragfähigkeit des Konzepts in Bezug auf den Aufbau einer heterogenen Simulationsumgebung in den verschiedenen Bereichen der computergestützten Biochemie.

Danksagung

Die vorliegende Arbeit enstand zu wesentlichen Teilen während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Fraunhofer Institut für Graphische Datenverarbeitung (IGD) in Darmstadt von Januar 1998 bis Juni 1999 und am Fraunhofer Anwendungszentrum für Computergraphik in Chemie und Pharmazie in Frankfurt am Main von Juni 1999 bis Dezember 2003.

An dieser Stelle möchte ich den vielen Personen danken, ohne deren Untersützung die Vollendung dieser Arbeit nicht möglich gewesen wäre.

- An erster Stelle danke ich Herrn Prof. Dr.-Ing. Krömker, dessen Betreuung und Unterstützung diese Arbeit erst ermöglicht haben.
- Ich danke Herrn Prof. Schneider für die Bereitschaft zur Übernahme des Korreferats dieser Arbeit und für seine Unterstützung.
- Mein weiterer Dank gilt allen meinen Kollegen am Fraunhofer IGD und Fraunhofer AGC, insbesondere Prof. Ralf Dörner und Dr. Paul Grimm. Mein besonderer Dank gilt Jens Barthelmes, dessen Anmerkungen unersetzbar waren.
- Weiterhin möchte ich meinen Studien- und Diplomarbeitern danken: Rolf Theisinger, Jan-Patrik Schäfer, Paul Izquierdo Rojas, Steffen Hartmann, Boris Zerbst, Lidong Huang, und Jan Walburg. Mein Dank gilt ebenfalls meinen verschiedenen wissenschaftlichen Hilfskräften, hier möchte ich Matthias Pfeiffer hervorheben.
- Nicht zuletzt danke ich meinen Eltern für ihre andauernde Unstützung. Ohne sie wäre mir weder Studium noch Promotion möglich gewesen.

Groß-Gerau, im September 2005

Christian Seiler

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

1.1. Motivation

Triebfeder für diese Arbeit ist die moderne Biochemie als Anwendungsfall für Methoden der Informatik und in diesem Fall speziell für die Methoden der Computergraphik.

Die Biochemie, selbst eine junge Wissenschaft, hervorgegangen aus der Chemie und der Biologie [1], beschäftigt sich mit der Chemie des Lebens und erhält ihre Motivation ihrerseits vor allem aus Fragestellungen der Biologie. Hinzu kommen aber wichtige Aufgaben aus der pharmazeutischen Forschung und der Medizin, weshalb eine Abgrenzung und Zuordnung nicht selten schwer fällt. Entsprechend wurde zu diesem Wissenschaftskomplex der Begriff „Life Sciences“ also „Lebenswissenschaften“ geprägt [2], denn die Gemeinsamkeit liegt in der Betrachtung von Prozessen in lebenden Zellen oder zwischen den Zellen.

Die moderne Biochemie ist dabei selbst in einem gewissen Übergang begriffen. Die bisherige Forschung auf diesem Gebiet ist sehr stark experimentell geprägt, eine theoretische Biologie analog zur theoretischen Chemie gibt es nur in Ansätzen. Trotzdem wandelt sich die Wissenschaft hin zu einer stärkeren Einbindung theoretischer Ansätze. Dies wird forciert durch die Erkenntnis, dass die bestimmenden biochemischen Prozesse in der Regel nicht von einzelnen wichtigen Reaktionen oder einfachen Reaktionsketten geprägt werden, sondern von Netzwerken. Hierbei handelt es sich vor allem um chemische Reaktionsnetzwerke; abgegrenzt nach der Funktion der beschriebenen Prozesse, spricht man von metabolischen Netzwerken [3], Netzwerken, welche die Signalübertragung innerhalb der Zelle beschreiben (Signaltransduktion), genetischen Expressionsnetzwerken etc. Während Ausschnitte dieser Netzwerke weiterhin experimentell aufgeklärt und verstanden werden, lässt sich das zusammenhängende Bild zunehmend nur noch durch eine theoretisch geprägte Modellbildung fassen.

Die Modellbildung ist nicht nur eine Domäne der Reaktionsnetzwerke, sondern findet ihren Weg von der Modellierung einzelner Moleküle bis zur Beschreibung von Zellhaufen und der Zelldifferenzierung. Während sich die Modellierung von Molekülen mit dem Design neuer Moleküle anhand von funktionalen Vorgaben beschäftigt, steht bei der Modellierung der Zelldifferenzierung die Wechselwirkung zwischen mehreren Zellen im Vordergrund, wobei sich die Dynamik der jeweiligen Zellen wiederum zum Teil über Netzwerke beschreiben lässt. Neue Forschungsergebnisse [112] unterstreichen darüber hinaus die Tatsache, dass Moleküle, Zellen und Zellhaufen dreidimensionale Gebilde sind – eine Tatsache, die bei der Modellbildung berücksichtigt werden muss.

Die Mächtigkeit formaler Modelle zeigt sich in ihrer Nutzung für die Simulation des Molekülverhaltens, der Netzwerke oder des Zellverhaltens. Die Simulationsstudie als Mittel wissenschaftlicher Forschung ist damit auf verschiedenen Stufen der computergestützten Biochemie vertreten.

Der Einsatz der Simulation bildet ein modernes Bindeglied zwischen theoretischer und experimenteller Forschung. Hierbei entstehen allerdings auch Probleme, denn weder die Simulationstechnik noch das Erstellen von Softwarewerkzeugen liegt im Schwerpunkt der Biochemie. Auch die etablierte Disziplin der Bioinformatik fokussiert bisher eher auf Speicherung, Analyse und Retrieval biologischer Daten als auf die Simulationstechnik. Entsprechend fehlt es an dedizierten, angemessenen Werkzeugen für diese Aufgaben.

Aus diesen Ausführungen lässt sich die Rolle, welche die Informatik und besonders die Computergraphik in diesem Szenario spielen muss, wie folgt beschreiben: Im Sinne eines

Dienstleisters ist es Ziel dieser Arbeit, Konzepte und Lösungen als Werkzeuge für die Forschung in der Anwendungswissenschaft zu liefern. Die Anforderungen an einen Forscher als „Werkzeuglieferanten“ unterscheiden sich hierbei von den Anforderungen einer „reinen“ Forschung, also Forschung allein in der Kerndisziplin der Informatik bzw. der Computergraphik.

Die Problemstellung aus dem Anwendungsgebiet ignoriert in der Regel Technologieentwicklungen und Unterdisziplinen der Informatik, wie etwa Simulationstechnik, Visualisierung oder Virtual Reality. Sie verlangt vielmehr nach „Querschnittslösungen“, also Lösungen, bei denen nicht *eine* bestimmte Technologie im Vordergrund steht, sondern solche, die in der Regel verschiedene Technologien *angemessen* verbinden.

Die oben beschriebenen Problemstellungen der computergestützten Biochemie werden für diese Arbeit in drei Informatik-Disziplinen übersetzt:

- Human-Computer-Interaction (HCI), also dedizierte Visualisierungen, Ein-/ Ausgabemetaphern, Interaktion und Kommunikation
- Simulationstechnik
- Softwareengineering/Systemarchitektur des Gesamtsystems.

Human-Computer-Interaction als Teildisziplin der Computergraphik beschäftigt sich mit dem Design von Mensch-Maschine-Schnittstellen. Ausgehend von der Prämisse, dass es keine für *alle* Anwendungen *optimale* Benutzungsschnittstelle gibt, ist es Ziel dieser Arbeit, für das ausgewählte Anwendungsfeld und damit verbundene spezifische Anwendungsfälle eine geeignete Lösung zu entwickeln.

Die Simulationstechnik, „Angelpunkt“ der computergestützten Systembiologie, ist ein heterogenes Forschungsfeld, das selbst nicht im Fokus der Computergraphik liegt. Für den Bereich der computergestützten Biologie haben sich hier aber einige Simulationstechniken für verschiedene spezifische Problemfälle herausgebildet. Ziel dieser Arbeit ist demnach, ein Simulationskonzept zu entwerfen, das es erlaubt, heterogene Simulationstechniken zu verbinden und so zu erweiterten Simulationsstudien zu gelangen. Zusätzlich werden gegenwärtig die Möglichkeiten dieser Simulationstechniken eingeschränkt, weil unterstützende Werkzeuge fehlen. Ziel muss es also zusätzlich sein, ein dediziertes, d. h. ein auf das Anwendungsgebiet abgestimmtes Simulationssystem zu entwerfen, das gleichzeitig offen und erweiterbar bleibt. Offenheit und Erweiterbarkeit sind deshalb von Bedeutung, weil flexibel auf neue Simulationstechniken eingegangen werden muss und die Integration unterstützender Werkzeuge möglichst einfach sein soll.

Schließlich ist es notwendig, die beschriebenen Teilsysteme in ein Gesamtsystem zu integrieren. Dieses System muss flexibel genug ausgelegt sein, um auch den Austausch eines Teilsystems zu ermöglichen, ohne die Validität des restlichen Teilsystems zu verletzen.

Die gegenwärtig vor allem in der Systembiologie verwendeten Modelle offenbaren ein besonderes Manko der bis jetzt zur Verfügung stehenden Werkzeuge: Räumliche Phänomene lassen sich bis jetzt nur schlecht abbilden. Die im Rahmen dieser Arbeit zu erstellenden Konzepte und Werkzeuge sollen daher insbesondere der dreidimensionalen Natur der Forschungsgegenstände der Biochemie (von Molekülen bis zu Zellverbänden) Rechnung tragen. Hieraus ergeben sich die Verbindungen zwischen den oben beschriebenen Teildisziplinen sowie die herausgehobene Rolle, die die Computergraphik in dieser Arbeit spielt.

Die vorliegende Arbeit präsentiert entsprechend für den Bereich der Benutzungsschnittstelle eine dedizierte anwendungsangepasste Ein-/ Ausgabemetapher, die *Virtual Glove Box*. Für die

Bereiche der Simulationstechnik und Systemarchitektur wird ein integriertes Konzept zur agentenbasierten 3-D-Simulation, verbunden mit der Kopplung heterogener Simulatoren, in einem Gesamtsystem beschrieben.

1.2. Aufbau der Arbeit

Diese Arbeit gliedert sich in fünf Hauptkapitel. Nach dieser Einführung widmet sich das zweite Kapitel den grundlegenden Erkenntnissen, auf denen die im Weiteren vorgestellten Konzepte und Lösungen aufbauen. Zunächst wird dabei das Konstrukt des *Intelligenten Software-Agenten* eingeführt und besprochen. Hierbei liegt ein besonderer Schwerpunkt auf der Beschreibung von Multi-Agenten-Systemen und dem Stand der Forschung im Bereich der Agentenkommunikation.

Der Einführung der *Intelligenten Software-Agenten* schließt sich ein Kapitel zur *Simulation* als wissenschaftlicher Methode der Erkenntnisgewinnung an. In diesem Kapitel werden die wesentlichen Technologien, wie etwa Methoden der ereignisdiskreten Simulation oder der zeitkontinuierlichen Simulation, vorgestellt und kurz diskutiert.

Die nächsten beiden Unterkapitel wenden sich nun der Computergraphik zu. Es wird zunächst ausschnittsweise die *Visualisierung* als computergraphische Methode zur Darstellung von Informationen beschrieben. Das Kapitel zur *Interaktion* beschäftigt sich mit der Schnittstelle zwischen Benutzer und Computer mit einem Schwerpunkt auf haptischer Interaktion, einer Technik, die für das spätere Interaktionskonzept von Bedeutung ist.

Das zweite Kapitel schließt ab mit einem Unterkapitel zu den Grundlagen des Anwendungsgebiets, der computergestützten Biochemie und einer Zusammenfassung. In diesem Kapitel wird ebenfalls der gewählte Ausschnitt aus dem Anwendungsbereich präzisiert, in dem die Themen *Molecular Modeling*, *Systembiologie* und *Zelldifferenzierung* hervorgehoben werden, eine Thementrias, die auch im Weiteren immer wieder aufgegriffen wird.

Im dritten Kapitel wird das Anwendungsgebiet in Hinblick auf Anforderungen untersucht. Dabei werden zunächst *Anforderungen* diskutiert, die an die *Simulation* in der computergestützten Biochemie zu stellen sind. Auf dieser Grundlage werden *Anforderungen an die Interaktivität und Visualisierung* formuliert. In diesen beiden Unterkapiteln werden Strukturen identifiziert, die auch über den Anwendungsfall der computergestützten Biochemie hinaus Gültigkeit haben. Die sich hieraus ergebenden Anforderungen werden dargestellt.

Kapitel vier stellt die zentralen Konzepte dieser Arbeit vor. Es handelt sich dabei zunächst um ein Konzept zur Kopplung heterogener Simulatoren mit Hilfe der Agententechnologie. Weiterhin wird ein Konzept zur Visualisierung und zur Interaktion vorgestellt. Diese Konzepte nehmen die strukturellen Überlegungen des Simulationskonzepts auf, so dass darauf folgend eine Integration der Konzepte gelingt.

Das fünfte Kapitel beschreibt die Realisierungen, die auf Grundlage der bis dato dargestellten Konzepte implementiert wurden. Nachdem zunächst das realisierte *Hardwaresystem* dokumentiert wird, liegt der Schwerpunkt dann auf den erstellten Softwaresystemen. Hierbei werden für alle drei identifizierten Teilbereiche der computergestützten Biochemie entsprechende Systeme beschrieben. Analog zu den Anforderungsanalysen im dritten Kapitel beschäftigt sich auch das fünfte Kapitel mit Realisierungen jenseits der computergestützten Biochemie, die die Konzeptbildung dieser Arbeit beeinflusst haben oder die nach Prinzipien erstellt wurden, die den in dieser Arbeit vorgestellten Konzepten ähnlich sind.

Kapitel sechs fasst die im vierten und fünften Kapitel vorgestellten Konzepte und Realisierungen noch einmal zusammen, stellt sie den Anforderungen gegenüber und evaluiert

sie. Dabei werden noch offene Punkte identifiziert, die im Rahmen dieser Arbeit noch keiner Lösung zugeführt werden konnten. In Kapitel sieben wird eine *Zusammenfassung* der gesamten Arbeit präsentiert und ein *Ausblick* auf zukünftige Arbeiten und Entwicklungen gegeben. Literaturverzeichnis und Anhänge bilden den Abschluss der Arbeit.

2. Grundlagen

Im folgenden Kapitel werden die Grundlagen beschrieben, die im weiteren Verlauf der Arbeit insbesondere für die Beschreibung des eigenen Konzepts wichtig sind. Es werden hierbei fünf Bereiche genauer beleuchtet:

- Intelligente Software-Agenten
- Simulation
- Visualisierung
- Mensch-Maschine-Interaktion
- Anwendungsgebiet Computational Biochemistry.

2.1. Intelligente Software-Agenten im Kontext von Visualisierung und Simulation

Zuerst wird der Begriff der *Entität* motiviert und umrissen. Daraufhin wird auf die Theorie der Intelligenten Software-Agenten eingegangen. Es werden besonders die Probleme der Multi-Agenten-Systeme und die Kommunikation zwischen Agenten beleuchtet.

3-D-computergraphische Anwendungen enthalten sehr oft gegenständliche oder abstrakte Darstellungen abgrenzbarer Objekte. Diese Objekte müssen dabei nicht notwendigerweise Gegenstände der realen Welt, wie Wände, Tische, Stühle etc., beschreiben; sie können z. B. auch stellare Objekte auf der einen Seite oder subatomare Strukturen auf der anderen Seite darstellen. Die Art, wie solche Objekte dargestellt werden, ist eine Fragestellung der Visualisierung und wird später behandelt. Diese Arbeit fokussiert auf solche 3-D-Objekte, die neben Geometrie und Aussehen auch ein *Verhalten* haben. Solche Objekte werden im Folgenden *Entitäten* genannt.

Als Beispiele für Entitäten seien hier virtuelle Personen, virtuelle Gegenstände oder virtuelle Atome genannt. Die Verhaltensbeschreibung erfolgt dabei auf unterschiedlichen Ebenen, wobei die folgenden Unterteilungen gewählt wurden [4]:

- geometrisches Verhalten
- simulatives Verhalten
- Aufgabenbeschreibung.

Auf der untersten Ebene erfolgt die Beschreibung des geometrischen Verhaltens. Hier werden also die geometrischen Bausteine definiert, aus denen die Entität aufgebaut wird. Zusätzlich werden hier die Abhängigkeiten zwischen diesen Bausteinen beschrieben, wie z. B. Veränderungen von Winkelstellungen, wie sie bei der Animation von kinematischen Ketten auftreten. Im Bereich der Animation virtueller menschlicher Körper existieren für diesen Bereich verschiedene Bibliotheken (z. B. [186]). Über solche Bibliotheken werden die „Bausteine“ eines Menschen definiert (grob: „Fuß“, „Unterschenkel“, „Oberschenkel“ etc.) sowie die Verbindung (Gelenke) zwischen diesen Bausteinen. Um diese Verhaltensbeschreibung aus einer höheren Ebene heraus nutzen zu können, ist es wichtig, die Schnittstelle klar zu definieren. Das bedeutet, dass die Freiheitsgrade klar beschrieben werden müssen.

Auf der nächsten Ebenen, der Ebene des simulativen Verhaltens, werden die Bausteine der darunterliegenden Ebene gesteuert. Das Verhalten von Entitäten kann durch ein Modell

beschrieben werden, welches mittels einer Simulation umgesetzt wird. Ein sehr einfaches Beispiel ist ein Ball (im einfachsten Fall besteht er aus einem einzigen geometrischen Baustein), dessen Verhalten durch eine physikalische Beschreibung definiert ist. Aus der Simulationsebene heraus muss dafür in der darunterliegenden geometrischen Ebene auf die Position und auf die Verformungsmatrix zugegriffen werden. Die Fragestellungen der Simulation werden im nächsten Kapitel beleuchtet.

Als oberste Ebene wird die Ebene der aufgabenorientierten Beschreibung von Entitäten gesehen. Als Beispiel hierfür sei die Beschreibung einer Person genannt, die von einem virtuellen Raum zuerst in einen Gang gehen soll, um dann die Tür eines weiteren Raums zu öffnen und einzutreten. Die Aufgabe „den Gang hinuntergehen“ wird durch Nutzung der simulativen Ebene („gehen“) erfüllt. Die Bausteine dieser Ebene sind also „Aufgaben“, die Schnittstellen zur Verfügung stellen müssen, damit diese verknüpft werden können.

Damit der Autor bei der Verhaltensbeschreibung auf einer Ebene auf die Verhaltensbeschreibungen der darunterliegenden Ebenen zugreifen kann, ist eine Aufteilung der Verhaltensbeschreibung notwendig. Diese Anforderung kann mit Hilfe der Theorie der Intelligenten Software-Agenten erfüllt werden.

Intelligente Agenten sind dabei als eigenständige Softwareeinheiten zu verstehen, deren typische Eigenschaften im Folgenden genauer betrachtet werden sollen (nach [11]):

- Autonomie
  Ein Agent handelt autonom, d. h., er ist bei seiner Tätigkeit nicht von ständigen Eingaben oder Kontrollen eines Benutzers abhängig. Er hat selbst die Kontrolle über seine inneren Zustände und seine Aktionen.

- Reaktivität
  Der Agent nimmt seine Umgebung wahr und reagiert auf äußere Reize. Die Umgebung eines Agenten muss nicht notwendigerweise die reale physische Welt sein, sondern kann beispielsweise auch ein im Computer modelliertes Szenario sein.

- Proaktivität
  Eine rein reaktive Haltung zur Umgebung schränkt die Möglichkeiten eines Agenten stark ein, denn Verhaltensweisen, die darauf beruhen, dass ein Agent selbst die Initiative ergreift, sind nicht modellierbar. Der Agent muss daher neben der Fähigkeit zur Reaktion auch die Möglichkeit haben, von sich aus aktiv zu werden.

- Schlussfolgerungsfähigkeit
  Eng mit der Möglichkeit, von sich aus aktiv zu werden, ist auch die Fähigkeit zur Schlussfolgerung verbunden. Ein Agent trifft seine Entscheidungen aufgrund seiner vorherigen Erfahrungen, er hat also ein Gedächtnis und kann entsprechend Schlüsse ziehen. Was er sich merkt und welche Folgerungen ein Agent zieht, hängt von dem zugrunde liegenden System ab, welches sein Verhalten beschreibt.

- Kommunikationsfähigkeit
  Schlussfolgerungen und Entscheidungen haben nur dann einen Sinn, wenn der Agent auch entsprechend auf seine Umwelt einwirken kann. Zur Kommunikationsfähigkeit gehört nicht nur die Kommunikation mit anderen Agenten, sondern auch die Kommunikation mit menschlichen Benutzern in der Umgebung des Agenten.

- Persönlichkeit
  Immer dann, wenn Agenten Menschen vertreten, wie beispielsweise in einem Erzählkontext, wird eine Glaubwürdigkeit erst dann erreicht werden, wenn ein Agent eine gewisse Persönlichkeit besitzt. Für einen spannenden Erzählverlauf mag es

beispielsweise hilfreich sein, wenn ein Agent nicht stereotyp und damit leicht vorhersagbar, sondern individuell reagiert.

Agentenorientiertes Programmieren ist ein neuer Trend der Software-Entwicklung in den letzten zehn Jahren – eine Technik, die eigene Methoden für die Spezifikation, Implementierung und Validierung hervorgebracht hat. [5]. Diese Technologie zielt besonders auf den Einsatz zur Lösung komplexer, oft verteilter Probleme [6].

Die historische Entwicklung der Programmiermethodik verlief von der Erstellung monolithischer Programme über strukturiertes Programmieren mit Prozeduren und Unterprogrammen zu den gegenwärtig aktuellen Methoden der objektorientierten Programmierung [7]. Die Weiterentwicklung der gekapselten Objekte, die von anderen Objekten erben können [8], führt schließlich nach dem Ansatz dieser Arbeit zu dem bereits vorgestellten Agentenparadigma der autonomen, reaktiven, proaktiven, sozialen und kommunikativen Softwareeinheiten mit eigener Kontrolle über ihren Programmfluss sowie internen Zuständen und Zielen.

Auf Basis des Agentenparadigmas bildete sich über die Zeit eine Reihe verschiedener Agenten-Architekturen, für die jeweils eine theoretische Grundlage entwickelt wurde.

2.1.1 Agenten-Architekturen

Es wurde eine Reihe verschiedener Theorien und Architekturen entwickelt, wie die konzeptionellen Vorgaben des Agentenparadigmas in Softwaretechnik umgesetzt werden können. Im Folgenden sollen einige der wichtigsten kurz skizziert werden.

Agenten-Architekturen bilden die Brücke von Agenten-Theorien zur Implementation von Agentensystemen. Dabei hat sich eine Reihe von grundsätzlichen Architekturen herausgebildet.

Die einfachste Agenten-Architektur ist die der so genannten *Reaktiven Agenten*. Reaktive Agenten basieren auf der von Brooks [9] formulierten Annahme des „reasoning without representation“. Derartige Agenten haben kein internes Abbild der sie umgebenden Welt und (in vielen Fällen) keinen inneren Zustand. Die Entscheidungen dieser Agenten sind situativ und nur von gegenwärtigen Sensorinformationen abhängig.

Das andere Extrem der Agenten-Architekturen sind so genannte *Deliberative Agenten*. Die Bezeichnung „deliberativ“ bezieht sich auf eine Architektur, die auf den Hypothesen von Newell und Simon [10] aufbaut. Demnach lässt sich jegliche intelligente Aktivität auf die Manipulation von Symbolmustern zurückführen. Die Symbole bilden hierbei ein Modell des Problembereichs (zu Modellbildung und Systemen siehe auch Kapitel 3.2 „Simulation“). Diese Manipulationen sind geeignet, aus gegebenen Mustern neue zu erzeugen und damit mögliche Lösungen des Problems zu generieren. Darüber hinaus muss es Mechanismen geben, die auf den Symbolmustern Suchfunktionen realisieren, um so geeignete Lösungen zu selektieren.

Hieraus ergibt sich die Notwendigkeit für Agenten, ein explizites Modell ihrer Umgebung im Speicher zu halten und zu verwalten. Als Methode der Verwaltung und Manipulation dieses Modells wurde von Rao et. al. [11] das Modell des *BDI*-Agenten entworfen. *BDI* steht hier für *Believes*, *Desires* und *Intentions*.

- *Believes* beschreiben das interne Modell der Umgebung des Agenten. Umgebung muss hierbei nicht notwendigerweise die physikalische Umgebung bedeuten (dies wäre etwa bei einem autonomen Roboter der Fall), sondern kann für beliebige „Problemfelder“ stehen, in welchen sich der Agent „bewegt“. Agenten sind nicht *per definitionem* allwissend. Dies hat zur Folge, dass es sich bei den *Believes* keineswegs um ein

korrektes Modell der Umgebung handeln muss, sie spiegeln lediglich den Zustand des Systems wieder, an dessen Korrektheit der Agent „glaubt“.

- *Desires* stehen für den angestrebten Zustand der Umgebung, die der Agent durch seine Aktionen erreichen will.
- *Intentions* geben den aktuell gewählten Aktionsplan vom durch die *Believes* beschriebenen aktuellen Umgebungszustand zum durch *Desires* ausgedrückten erwünschten Zustand wieder.

Der beschriebene BDI-Ansatz gibt jedoch noch recht wenige Hinweise auf die tatsächliche Umsetzung der Architektur. [11] beschreibt für BDI-Agenten eine Implementierung, die von der Annahme ausgeht, dass sich die Dynamik des Agenten und seiner Umwelt als Folge von Zuständen beschreiben lässt. Zu jedem Zeitpunkt gibt es eine große Anzahl von möglichen Veränderungen der Umwelt sowie eine Vielzahl von Möglichkeiten des Agenten zu agieren. Rao und Georgeff beschreiben dieses System durch einen Entscheidungsbaum. Die Knoten dieses Baums stellen Umweltzustände dar, während die Kanten entweder Aktionen des Agenten oder äußere Ereignisse darstellen. Jeder Pfad von der Wurzel des Baums zu einem Knoten bezeichnet einen Ausführungspfad. Gesucht sind demnach Pfade, die zu einem der durch *Desires* definierten Zielzustände führt. Die Schwierigkeit liegt nun darin, dass diejenigen Zustandsübergänge, die nicht auf Aktionen des Agenten zurückgehen, eben nicht in dessen Kontrolle liegen. Rao und Georgeff unterscheiden die Zustandsknoten des Baums entsprechend in zwei Kategorien:

- Choice nodes
- Chance nodes.

Choice nodes entstehen hierbei durch die Aktionen des Agenten.

Eine mögliche Lösung dieses Szenarios bietet die so genannte „Mögliche-Welten“-Semantik. Hierbei werden aus einem Baum mit *Chance nodes* mehrere Bäume, aus denen genau diese Knoten eliminiert wurden (siehe Abbildung 1). Jeder dieser Bäume beschreibt eine „mögliche Welt“.

Die eliminierten *Chance nodes* werden dabei mit Werten belegt, die die Wahrscheinlichkeit ihres Eintreffens beschreiben. Nach der Elimination der Knoten wird die kumulierte Wahrscheinlichkeit dem nun unabhängigen Baum zugeordnet. Zusammen mit einer Bewertung der Endzustände im Vergleich zu den *Desires* ergeben diese Bäume eine Grundlage für die Entscheidungen des Agenten.

Hybride Agenten-Architekturen versuchen, die verschiedenen Ansätze miteinander zu verbinden. Einige Beispiele sind etwa das Procedural-Reasoning-System (PRS) [12], COSY [13] oder INTERRAP [14]. Eine Variante dieser hybriden Architekturen bilden *Schichten*-Architekturen (*Layered* Architektures) (siehe Abbildung 2).

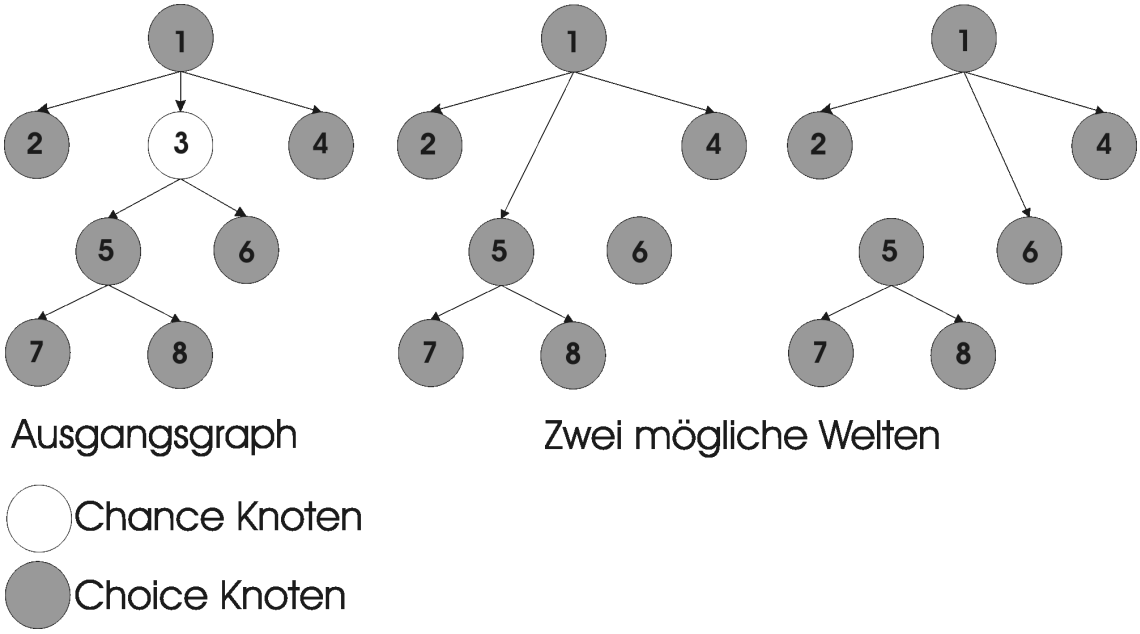


Abbildung 1 Mögliche Welten, nach [15]]

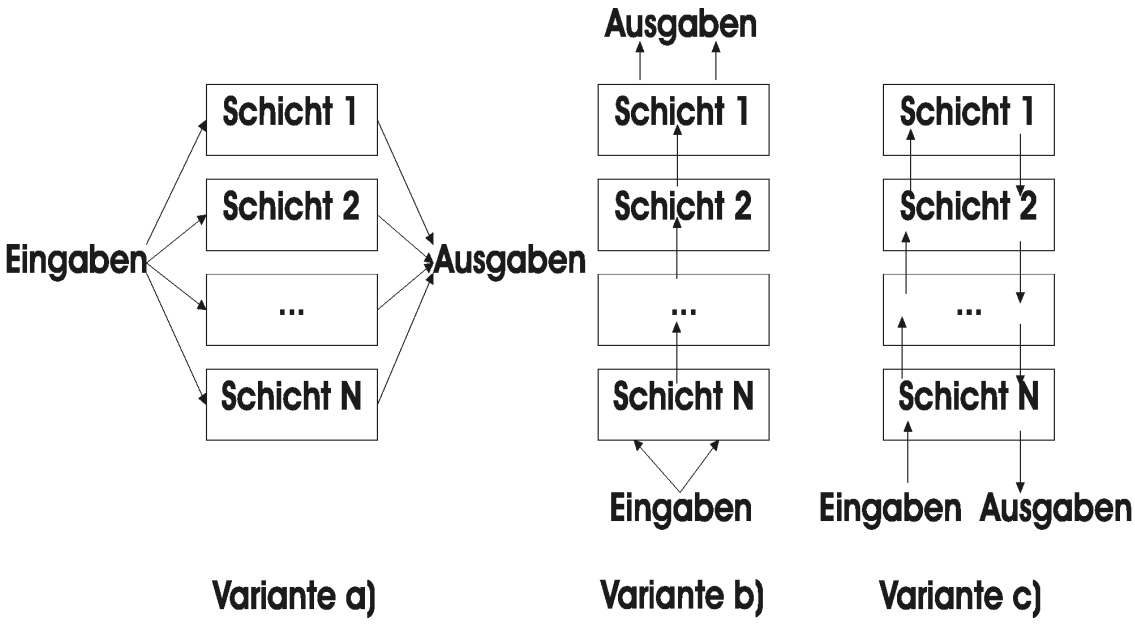


Abbildung 2 Schichtenarchitekturen

In dieser Schichtenarchitektur werden sensorische Eingaben von einer Schicht an weitere unabhängige Schichten weitergeleitet. Die Abbildung 2 zeigt verschiedene Varianten:

- Variante a) zeigt eine horizontale Architektur. Motivation ist hier, dass bestimmten Eingaben dedizierte Bearbeitungsschichten zugeordnet werden.
- Variante b) beschreibt ein kaskadenförmiges System, in dem eine Schicht jeweils als Filter oder Präprozessorstufe für die nächste Stufe fungiert.
- Variante c) zeigt schließlich ein Unterprogrammsystem. Hier fungiert eine Schicht als Selektionsstufe, welche die Dienste der nächsten Stufe in Anspruch nimmt, um letztendlich Ausgaben zu erzeugen.

2.1.2 Multi-Agenten-Systeme

Der Einsatz von intelligenten Agenten ist vielfältig und daher vielfacher Gegenstand der Forschung. In einer Reihe von Einsatzgebieten werden Agenten als Einzelentitäten genutzt. Hierzu gehören beispielsweise Interface-Agenten, die dem Nutzer individualisierte Schnittstellen zur Nutzung weitergehender Applikationen bieten, oder etwa Robotersteuerungen [15]. Darüber hinaus existiert eine Anzahl von Anwendungen, in denen mehrere Agenten gleichzeitig und kollaborativ in einer gemeinsamen Umgebung eingesetzt werden. Für derartige Anwendungen hat sich der Ausdruck *Multi-Agent-Systems (MAS)* geprägt. Im Folgenden sollen einige wichtige Aspekte dieser Gattung von Agentensystemen betrachtet werden.

Ausgehend von den allgemeinen Agentensystemen, wie sie oben beschrieben wurden, hat die Forschung Systeme entwickelt, die Elemente der verteilten Systeme mit denen der Agenten-Theorie verbinden [16], [17], [18].

Für das Design von Multi-Agenten-Systemen gibt es bisher noch kein allgemein gültiges formales Framework, sondern lediglich durch das jeweilige Anwendungsfeld bestimmte Insellösungen. Dennoch existieren Gemeinsamkeiten, die Multi-Agenten-Systeme von konventionellen Softwaresystemen unterscheiden [19]. In allen MAS müssen Fragen der Zuständigkeiten und der Zieldefinition der einzelnen Agenten gelöst werden [20], [21]. Fragen wie diese verbinden die Anwendungsdomäne mit der Entwicklung der einzelnen Softwarekomponenten und dem Design des Multi-Agent-Systems insgesamt. Darüber hinaus stellen sich folgende Aufgaben [22]:

- Anwendungsbereich Analyse: Identifikation der verschiedenen Funktionen und Aufgaben der Anwendungsdomäne. Dies geschieht unter dem Gesichtspunkt der Aufteilung dieser Aufgaben.
- Design der (Einzel-)Agenten: Definition von Zielen, Zuweisung von Rollen mit Zuständigkeiten und Fähigkeiten, Definition des internen Wissens des Agenten (Regeln etc.)
- Design des Multi-Agenten-Systems: Definition der Kooperationsparameter und der Kommunikation. Design der Kontrollfunktionen und der Protokolle für die Konfliktbehandlung.

Bei Umsetzung dieser drei Punkte sind einzelne Softwarekomponenten (Agenten) in einem verteilten System (MAS) bereit, die Problemstellung (aus dem Anwendungsgebiet) kooperativ und kommunikativ zu bearbeiten.

Der zentrale Faktor eines Multi-Agenten-Systems ist demnach die Kooperations- und Kommunikationskomponente. Es haben sich drei Kooperationsvarianten in der Literatur herausgebildet:

- Kooperative Interaktion: Kooperative Interaktion steht für Systeme, in denen die Agenten sich gegenseitig bei der Lösung ihrer Aufgaben unterstützen. Die Koordination wird dabei implizit in die Architektur der einzelnen Agenten eingebaut, etwa durch die Definition von entsprechenden *Desires* oder durch besondere „Belohnungen“ [23]. Dies kann auch den Austausch von Zwischenergebnissen beinhalten [24] oder über die gemeinsame Nutzung von Ressourcen geregelt sein [25].
- Kontraktbasierte Kooperation: Bei dieser Kooperationsform erklärt ein Agent seine Bedürfnisse, und andere Agenten *verpflichten* sich zu Handlungen, zur Übergabe von Informationen etc. (*commitment*). Die Auswahl des Dienstanbieters findet dabei durch eine auktionsähnliche Verhandlung statt [26]. Einige der gebräuchlichsten Strukturen sind:
    - die verdeckte Auktion: Jeder Dienstanbieter gibt sein Gebot ab, ohne die Gebote der anderen Agenten zu kennen. Der „billigste“ Anbieter gewinnt. Die Kostenfunktion ist hierbei, wie auch bei den anderen Auktionsformen, anwendungsabhängig.
    - die Englische Auktion: Die Gebote sind öffentlich und werden sequenziell abgegeben. Jedes neue Gebot muss besser (billiger) als das bisher führende Gebot sein. Der Bieter des letzten Gebots erhält den Zuschlag.
    - die Holländische Auktion: Der Nachfrager beginnt die Auktion mit einem geringen Startpreis. Dieser Preis wird sukzessive erhöht, bis sich ein Anbieter findet.

  Der gebräuchlichste Ansatz basiert auf dem Ansatz der verdeckten Auktion (Contract-Net-Protocol) [27].
- Verhandlungsbasierte Kooperation: Dieser Ansatz geht von nebenläufigen Agenten aus, die immer dann miteinander verhandeln müssen, wenn mehrere Agenten gleichzeitig knappe Ressourcen nutzen wollen, um ihre Aufgabe zu erfüllen [28], [5]. Die Lösung dieser Konflikte kann von außen vorgegeben werden (Prioritätswarteschlangen, Zeitscheibenverfahren etc.) oder durch explizite Regeln in den Agenten ausformuliert werden.

In einem Einzel-Agenten-System agiert der Agent aufgrund seiner Regeln oder seiner Planungsarchitektur, um gesteckte Ziele zu erreichen. In einem Multi-Agenten-System müssen die einzelnen Agenten jedoch noch die Aktionen der anderen Agenten berücksichtigen. Der Schritt von einem Einzel-Agenten-System über ein System nebenläufiger Agenten, deren Kooperation sich im Wesentlichen durch die Auflösung von Konflikten bei knappen Ressourcen auszeichnet, zu *sozialen* Agenten vollzieht sich nach [29] durch das Prinzip der „sozialen Rationalität“: “Within an agent-based society, if a socially rational agent can perform an action so that agent’s joint benefit is greater than their joint loss then it may select that action”.

Der Wert einer Aktion *a, W(a),* lässt sich demnach durch eine Funktion wie in Gleichung 1 beschreiben:

$$W(a) = f(IW(a), SW(a))$$

Gleichung 1: Wert einer Aktion

Dabei ist *IW(x)* der individuelle Wert der Aktion *x*, *SW(x)* ist der soziale Wert der Funktion, und *f* ist die anwendungsspezifische Gewichtungsfunktion zwischen individueller und sozialer Bewertung. Hogg et. al. gehen in ihrer Beschreibung allerdings von allwissenden Agenten aus, nur dann ist eine objektive Bewertung des sozialen Aspekts der Aktion möglich. Verschiedene *Believes* über den Zustand der Welt und über den Zustand andere Agenten führen zu Phänomenen, wie sie etwa Gegenstand der Spieltheorie sind [30].

2.1.3 Agentenkommunikationssprachen

Oben wurde *Kommunikationsfähigkeit* als eine der Eigenschaften genannt, die Agenten charakterisieren. Die Fähigkeit ist abhängig vom jeweiligen Kontext der Anwendung des Agentensystems. Ausgehend von der Idee generischer Multi-Agenten-Systeme ist auch die allgemeine Kommunikation zwischen den Agenten Gegenstand der Forschung. Ergebnisse dieser Forschung sind so genannte Agentenkommunikationssprachen (*Agent Communication Languages, ACL*). Der Aufbau dieser Sprachen gründet sich auf der Theorie natürlicher Sprachen, der *Sprachtheorie* [31].

Grundlage der heute gängigsten Agentenkommunikationssprachen sind Werke aus den 1960er Jahren von Austin [31] und später Searle [32]. Beide beschäftigten sich mit Möglichkeiten, natürliche Sprache zu klassifizieren und formal zu fassen. Sie erkannten in den Äußerungen natürlicher Sprache so genannte „Sprachakte“ (*Speech Acts*).

Die Grundannahme von Austin, später erweitert und verfeinert von Searle, ist, dass jeder Mensch etwas *tut*, wenn er etwas *sagt*. Ein Beispiel mag dies verdeutlichen: Wenn ein Sprecher sagt „Ich verspreche, dass *p* gilt.“, ist dies keine Aussage im Sinne der Aussagenlogik. Dies bedeutet, dass eine solche Äußerung nicht *wahr* oder *falsch* sein kann, sondern lediglich *angebracht (felicitous)* oder *unangebracht (infelicitous)*. Austin nennt solche Äußerungen *Ausführungen*. Hier hat sich im Sprachgebrauch der englische Ausdruck *performatives* (von engl. „to perform“ = "ausführen“) durchgesetzt.

Austin unterscheidet Sprachakte in *locutionary acts, illocutionary acts und perlocutionary acts*.

- *Locutionary acts* sind Sprechakte, die lediglich eine Feststellung beinhalten.
- *Ilocutionaty acts* sind Sprachakte, die etwas *bewirken* können, wie etwa ein Befehl, eine Warnung etc.
- *Perlocutionary acts* sind schließlich Akte, die etwas ausgelöst haben, die etwa überraschend oder überzeugend sind.

Searle führt dazu aus, dass sich praktisch alle tatsächlichen Äußerungen als illocutionary acts beschreiben lassen. Er beschreibt dabei fünf Klassen von performatives innerhalb der illocutionary acts:

- *Assertives:* Ausdrücke, die über den Zustand der Welt (in der Regel: von Teilen der Welt) Auskunft geben. Solche Ausdrücke lassen sich aussagenlogisch als *wahr* oder *falsch* bewerten.
- *Directives:* Äußerungen, die den Hörer zu etwas bewegen sollen, also etwa Befehle, Anfragen, Verbote etc.
- *Commisives:* Äußerungen, in denen sich der Sprecher zu etwas verpflichtet.
- *Expressives:* Äußerungen, die den (emotionalen) Zustand des Sprechers bezüglich des Sprechakts ausdrücken, so z. B. Entschuldigungen, Beschwerden, Lob etc.

- *Declaratives:* Äußerungen, die durch den Sprechakt selbst den Zustand der Welt verändern (sollen), etwa Ernennungen, Bestätigungen usw.

Obwohl sich die Erkenntnisse der *Speech Act Theory* zunächst lediglich auf natürliche Sprache bezogen, wurden sie zur wichtigsten Grundlage der Agentenkommunikationssprachen.

Die wichtigsten Vertreter dieser Agentenkommunikationssprachen sind die KQML, die Knowledge Query and Manipulation Language [33] und die FIPA Communicative Act Library Specification der *Foundation for Intelligent Physical Agents* [34].

2.1.3.1 KQML

KQML ist sowohl ein Nachrichtenformat als auch ein Nachrichtenaustauschprotokoll. Ziel ist der Austausch von Wissen (*knowledge sharing*) zwischen kooperativen Agenten. KQML nutzt ein Schichtenmodell der Kommunikation. Auf den unteren Schichten wird der eigentliche Nachrichtentransportmechanismus geregelt, während in den höheren Schichten die inhaltliche Kommunikation definiert ist. Entsprechend stellt KQML eine standardisierte Syntax für Nachrichten bereit, die auf den oben beschriebenen *performatives* der *Speech Act Theory* beruht. Die Performatives umfassen typische Konstrukte wie „tell“, „perform“, „reply“ etc. (siehe Tabelle 1).

Name	*Section*	*Meaning*
achieve	5.6	S wants R to do make something true of their environment
advertise	5.8	S is particulary-suited to processing a performative
ask-about	5.4	S wants all relevant sentences in R's VKB
ask-all	5.4	S wants all of R's answers to a question
ask-if	5.4	S wants to know if the sentence is in R's VKB
ask-one	5.4	S wants one of R's answers to a question
break	5.10	S wants R to break an established pipe
broadcast	5.10	S wants R to send a performative over all connections
broker-all	5.11	S wants R to collect all responses to a performative
broker-one	5.11	S wants R to get help in responding to a performative
deny	5.1	the embedded performative does not apply to S (anymore)
delete	5.2	S wants R to remove a ground sentence from its VKB
delete-all	5.2	S wants R to remove all matching sentences from its VKB
delete-one	5.2	S wants R to remove om matching sentence from its VKB
discard	5.7	S will not want R's remaining responses to a previous performative
eos	5.5	end of a stream of responses to an earlier query
error	5.3	S considers R's ealier message to be mal-formed
evaluate	5.4	S wants R to simplify the sentence
forward	5.10	S wants R to route a performative
generator	5.7	same as a standby of a stream-all
insert	5.2	S asks R to add content to its VKB
monitor	5.9	S wants updates to R's response to a stream-all
next	5.7	S wants R's next response to a previously-mentioned performative
pipe	5.10	S wants R to route all further performatives to a another agent
ready	5.7	S is ready to respond to R's previously-mentioned performative
recommend-all	5.11	S wants all names of agents who can respond to a performative
recommend-one	5.11	S wants the name of an agent who can respond to a performative
recruit-all	5.11	S wants R to get all suitable agents to respond to a performative
recruit-one	5.11	S wants R to get another agent to respond to a performative
register	5.10	S can deliver performatives to some named agent
reply	5.4	communicates an expected reply
rest	5.7	S wants R's remaining responses to a previously-mentioned performative
sorry	5.3	S cannot provide a more informative reply
standby	5.7	S wants R to be ready to respond to a performative
stream-about	5.5	multiple-response version of ask-about
stream-all	5.5	multiple-response version of ask-all
subscribe	5.9	S wants updates to R's response to a performative
tell	5.1	the sentence in S's VKB
transport-address	5.10	S associates symbolic name with transport address
unregister	5.10	a deny of a register
untell	5.1	the sentence is not in S's VKB

Tabelle 1 KQML-Performatives [33]

Der eigentliche Inhalt der Nachricht wird dabei nicht von KQML erfasst, es wird lediglich ein allgemeiner Rahmen geboten, mit dessen Hilfe die Kommunikation abgewickelt werden kann. KQML beschreibt eine asynchrone Kommunikation, die von Kommunikationskanälen ausgeht, die zwar die Reihenfolge der Nachrichten (-pakete) garantiert, nicht jedoch die verlustfreie Kommunikation.

2.1.3.2 KIF

Während die KQML das *Wie* der Kommunikation regelt, bleibt die Frage nach dem Inhalt der Nachrichten absichtlich unbeantwortet. Diese Aufgabe fällt dem *Knowledge Interchange Format (KIF)* zu. KIF beschreibt die Syntax des Nachrichteninhalts als Ausdrücke der

Prädikatenlogik erster Ordnung. Hierdurch sind Agenten in der Lage, den Inhalt von Nachrichten ohne zusätzliche Interpreter oder Filter zur verstehen und zu verarbeiten.

KIF ist vollständig in dem Sinne, dass es Mechanismen enthält, die es ermöglichen, beliebige gültige prädikatenlogische Ausdrücke zu formulieren. Darüber hinaus existieren Methoden für die Repräsentation von Wissen über die Repräsentation von Wissen, so genanntes Meta-Wissen bzw. Meta-Informationen. Das Format ist damit in der Lage, weitere deklarative Sprachen, wie etwa PROLOG, LISP, aber auch XML, einzubinden. Neben einer Syntaxdefinition für Nachrichten findet sich auch eine Definition für die Semantik. [35].

2.1.3.3 FIPA Communicative Act Library Specification

Obwohl die KQML eine starke Anhängerschaft gefunden hat, gab es auch einige Kritik, z. B. in [36]. Der wichtigste Punkt war dabei das Fehlen von *commisives,* also einer Klasse von Performatives, die es dem Agenten erlauben, sich zu einer Handlung zu verpflichten.

Auch in diesem Zusammenhang steht die seit 1995 andauernde Initiative der „Foundation For Intelligent Physical Agents“ (FIPA) für die Standardisierung einer Agentenkommunikationssprache. Das Ergebnis dieser Bemühungen ist die *FIPA Communicative Act Library Specification* [34]. Diese ACL ist der KQML sehr ähnlich und enthält folgende Performatives:

- *Accept Proposal*: Damit akzeptiert ein Agent ein vorher empfangenes *Proposal*.
- *Agree*: Mit dieser Aktion verpflichtet sich der Agent, eine Handlung vorzunehmen.
- *Cancel*: Der Sender informiert den Empfänger, dass eine bestimmte Aktion vom Empfänger nicht länger durchgeführt werden soll.
- *Call for Proposal*: Mit dem Call for Proposal wird eine Verhandlung initiiert mit dem Ziel, eine bestimmte Handlung durchzuführen. (Zum Protokoll der Verhandlung siehe den Abschnitt über *kontraktbasierte Kooperation* weiter oben in diesem Kapitel.)
- *Confirm*: Der Sender bestätigt dem Empfänger den Wahrheitsgehalt einer Aussage. (Dies geschieht, wenn der Sender weiß, dass der Empfänger über den Wahrheitsgehalt der Aussage unsicher ist.)
- *Disconfirm*: Der Sender informiert den Empfänger, dass eine Aussage *nicht* wahr ist. (Dies geschieht, wenn der Sender weiß, dass der Empfänger über den Wahrheitsgehalt der Aussage unsicher ist.)
- *Failure*: Damit teilt der Agent einem Empfänger mit, dass eine Handlung gescheitert ist.
- *Inform*: Der Sender informiert den Empfänger, dass eine Aussage wahr ist.
- *Inform If*: Eine zusammengesetzte Aktion. Im Prinzip ist dies ein *inform*, das inhaltlich davon abhängt, ob der Agent an die Wahrheit einer bestimmten Aussage glaubt oder nicht.
- *Inform Ref*: Dies ist ein Makro, mit dem eine Information über eine Referenz gegeben wird. Ein Beispiel für eine Referenz könnte sein: „Name des Autors“ mit dem Wert „Christian Seiler“.
- *Not Understood*: Das ACL-Äquivalent zu “Wie bitte?”

- *Propagate*: Der Empfänger soll eine in dieser Nachricht gekapselte Nachricht empfangen, als wäre sie direkt geschickt worden, außerdem soll die (auf bestimmte Art modifizierte) Nachricht an einen ausgesuchten Empfängerkreis weitergeschickt werden.
- *Propose*: Der Sender macht einen Vorschlag, eine bestimmte Aktion unter bestimmten Voraussetzungen durchzuführen.
- *Proxy*: Der Sender bittet den Empfänger *a*, eine Nachricht an einen zweiten Empfänger *b* weiterzuleiten.
- *Query If*: Mit deser Aktion stellt ein Agent die Frage, ob eine bestimmte Aussage wahr ist.
- *Query Ref*: Frage nach einer Menge von Aussagen, die einer gegebenen Referenz entsprechen, z. B. eine Datenbankabfrage.
- *Refuse*: Der Sender verkündet die Ablehnung, eine bestimmte Aktion durchzuführen (inklusive einer Erklärung hierfür).
- *Reject Proposal:* Ablehnung eines Proposals in einer Verhandlung.
- *Request*: Bitte des Senders an den Empfänger, eine bestimmte Aktion durchzuführen.
- *Request When*: Bitte des Senders an den Empfänger, einmalig eine bestimmte Aktion durchzuführen, wenn ein bestimmter Zustand eintritt.
- *Request Whenever*: Bitte des Senders an den Empfänger, eine bestimmte Aktion immer dann durchzuführen, wenn ein bestimmter Zustand eintritt.
- *Subscribe*: Die Bitte an den Empfänger, den Sender über den Wert einer Referenz zu informieren und dies zu wiederholen, wann immer sich der Wert ändert.

Auch die von der FIPA definierte Kommunikationssprache legt keine Beschreibungssprache für den Inhalt der Nachricht fest. Als *Informative Annex* wird allerdings eine *Semantic Language (SL)* definiert. Diese Sprache wurde auch benutzt, um die Semantik der Performatives im Standard zu definieren.

Die *SL* besteht aus prädikatenlogischen Ausdrücken, die um drei *mental attitudes* und vier Operatoren erweitert wurden. Die drei Haltungen sind:

- Believe: $B_i p$
  Agent *i* glaubt, dass p gilt.
- Uncertainty: $U_i p$
  Agent *i* ist sich nicht sicher, ob *p* gilt, glaubt aber, dass es wahrscheinlicher ist, dass *p* wahr ist, als dass *p* falsch ist.
- Choice: $C_i p$
  Agent *i* möchte, dass *p* jetzt wahr ist.

Die vier Operatoren sind:

- *Feasible(a,p)* bedeutet, dass die Aktion *a* durchführbar ist und dass *p* dann wahr wird.
- *Done(a,p)* bedeutet, dass Aktion *a* gerade durchgeführt wurde und dass *p* davor wahr war.

- *Agent(i,a)* bedeutet, dass Agent *i* der einzige Agent ist, der die Aktion (die Menge von Aktionen) *a* ausführen kann.
- *Single(a)* bedeutet, dass *a* eine einzige, nicht zusammengesetzte Aktion ist.

Für jedes Performative der ACL werden mit Hilfe der Semantic Language zwei Beschreibungen definiert:

1. *Feasibilitiy Preconditions (FP)*, also alle jene Voraussetzungen, die wahr sein müssen, damit der Agent den Sprechakt ausführen kann
2. *Rational Effect (RE)*, also der vom Agenten erwünschte Effekt seines Sprechaktes.

Im Anhang (Kapitel 9.1) findet sich ein Beispiel, wie Performatives in [34] definiert werden.

2.1.3.4 Ontologien

Die vorgestellten Agentenkommunikationssprachen sind allesamt mit dem Ziel definiert worden, generische Agentensysteme zu erstellen. Es hat sich jedoch gezeigt, dass für konkrete Anwendungen ein anwendungsgebietsspezifisches Vokabular definiert werden muss, um die semantischen Zusammenhänge eindeutig zu machen. In diesem Zusammenhang wurden so genannte *Ontologien* entwickelt [37], [38]. Für die Domäne des Wissensaustauschs steht die Bezeichnung "Ontologie“ für die abgeschlossene Beschreibung von Begriffen, Konzepten und deren Relationen zueinander, die für eine konkrete Gruppe von Agenten gelten. Diese Begriffe sind in einem formalen Vokabular niedergelegt.

Für eine Gruppe kommunizierender Agenten bedeutet dies, dass sich sämtliche Äußerungen innerhalb des gegebenen Kontextes bewegen müssen. Im Gegenzug ist garantiert, dass alle Agenten die entsprechenden Aussagen auf die gleiche Weise interpretieren. Zu einer Ontologie gehört eine Menge von Namen für Objekte und Objektklassen inklusive ihrer Attribute, ihrer Relationen zu anderen Objekten und Klassen sowie die für sie geltenden Bedingungen. Hinzu kommen Bedeutungen dieser Namen (dies kann auch informal, z. B. natürlichsprachlich gegeben sein) sowie Axiome, die für die betrachtete Problemdomäne gelten sollen.

Obwohl alle Agenten in einem Multi-Agenten-System im Prinzip das gleiche Vokabular haben, müssen sie nicht gleichartig auf eine gegebene Äußerung reagieren. Weiterhin müssen sie nicht die gleiche Wissensbasis besitzen, dies verbietet schon die Autonomiebedingung der oben genannten Agentendefinition.

2.1.4 Beispiele für Multi-Agenten-Systeme

Die später in dieser Arbeit vorgestellten Konzepte zur interaktiven Simulation und Visualisierung sind unabhängig von der konkreten Agentenimplementierung. Es existiert eine fast unüberschaubare Zahl von Agentenimplementierungen, deren komplette Darstellung weit über den Rahmen dieser Arbeit hinausgehen würde. Die folgenden Beispiele sollen den gegenwärtigen Stand von Multi-Agenten-Systemen demonstrieren. Ausgewählt wurden DECAF und FIPA-OS. DECAF ist ein im akademischen Bereich entwickeltes und vielfach eingesetztes System, während FIPA-OS neben einer starken Anbindung an die Entwicklungen der FIPA kommerzielle Wurzeln hat.

2.1.4.1 DECAF

DECAF (Akronym für *Distributed, Environment Centered Agent Framework*) ist ein an der University of Delaware in Newark, Delaware, USA entwickeltes *Toolkit*. Diese Sammlung von

Werkzeugen soll es erlauben, Multi-Agenten-Systemen unter einem Software-Engineering zu entwickeln. Das Toolkit bietet eine Plattform, auf der intelligente Software-Agenten im Sinne eines *Rapid Prototyping* [39] einfach und schnell erstellt und anschließend auch genutzt werden können. Die Plattform DECAF bietet die für ein Multi-Agenten-System notwendigen Basisdienstleistungen an. Diese Dienstleistungen umfassen:

- Kommunikation
- Planung
- Lernen
- Koordination
- Scheduling
- Performanz-Beobachtung
- Selbstdiagnose.

DECAF will damit so etwas wie ein „Betriebssystem“ für Multi-Agenten-Systeme sein, unter anderem mit dem Ziel, den Entwickler des Agentensystems von der Programmierung von Basisaufgaben zu befreien.

Ein DECAF-Agent wird über ein graphisches Interface-Werkzeug, genannt *Plan Editor*, programmiert. Die Grundbausteine innerhalb des Plan Editors sind ausführbare Aktionen, die in Form eines Netzwerks zu komplexen Aktionen verbunden werden können und schließlich ein hierarchisches Task-Netzwerk bilden (siehe Abbildung 3).

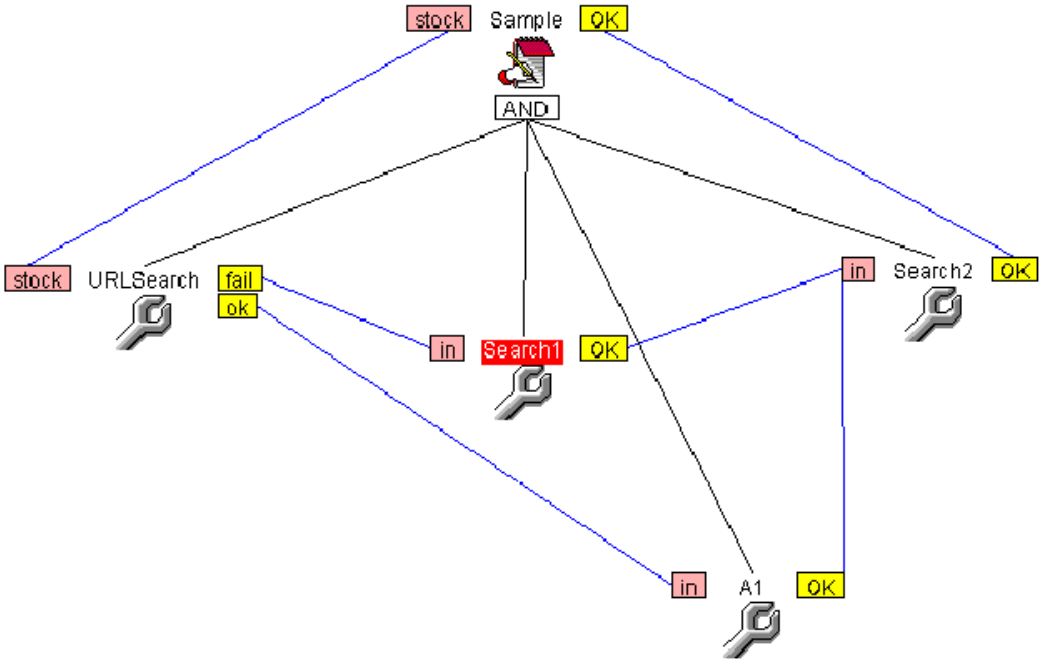


Abbildung 3 Beispiel eines DECAF-Plan [42]

Diese Architektur, basierend auf Komponenten und Frameworks, erlaubt die Wiederverwendung von Komponenten und eine Fehlerkontrolle bereits in der Designphase. Die Grundbausteine werden mit bekannten Konstrukten der imperativen Programmierung, wie *if-*

then-else oder Schleifen, verbunden. DECAF baut hierbei auf anderen Arbeiten wie RESTSINA [40] und TAEMS [41] auf.

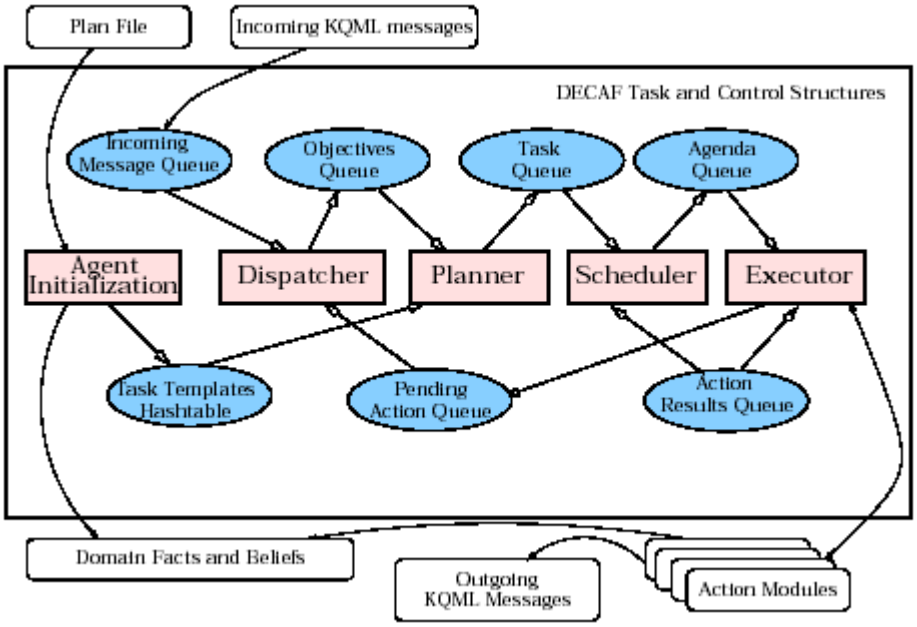


Abbildung 4 DECAF-Architektur [42]

DECAF erweitert den Komponentenansatz des Software-Engineering durch *Performance Profiles,* die einer elementaren Aktion zugeordnet werden können und die vom System für ein verbessertes Scheduling genutzt werden. Mit diesem Mechanismus lässt sich weiterhin die Wiederverwendbarkeit von Komponenten erhöhen, denn gesteuert durch das *Performance Profile* kann die Ausführung konkreter Aktionen in verschiedenen Ausführungsumgebungen verschieden organisiert sein. Es ist etwa möglich, dass ein Agent in einer Ausführungsumgebung nach einem optimalen Ergebnis sucht, während der gleiche Agent mit der gleichen Suchaktion in einer anderen Ausführungsumgebung das erste passende Resultat findet. So können verschiedene Design-Ansätze, wie etwa *Design-to-time* oder *Design-to-criteria*, [42] erprobt werden.

Abbildung 4 zeigt die verschiedenen Architekturkomponenten des in der Programmiersprache Java [43] implementieren DECAF-Systems. Der Fokus von DECAF liegt dabei darauf, die Entwicklung anwendungsspezifischer Agenten mit der Nutzung standardisierter Middle-Ware-Agenten transparent zu verbinden.

Der Agentenprogrammierer wird also von *Low-Level*-Aufgaben, wie etwa Socket-Kommunikation, dem Parsen eingehender Nachrichten und dem syntaktisch korrekten Verfassen von Nachrichten, entlastet. Damit verringert sich der Aufwand für den Programmierer, gleichzeitig wird eine Abstraktion von der tatsächlich verwendeten Netzwerk- und Kommunikationsinfrastruktur erreicht.

2.1.4.2 FIPA-OS

FIPA-OS wird von einem englischen Start-up-Unternehmen namens Emorphia entwickelt. Die Entwickler können dabei auf Erfahrungen in von der Europäischen Union geförderten Projekten und bei Nortel Technology, respektive Nortel Networks zurückgreifen. Bereits kurz nach der Veröffentlichung des ersten FIPA-Standards 1997 wurde FIPA-OS als weltweit erstes „FIPA-compliant"-Produkt, damals noch von Nortel Networks, veröffentlicht. FIPA-OS ist ebenfalls ein komponentenbasiertes *Toolkit* mit dem Ziel, schnell FIPA-kompatible Agenten zu entwickeln. FIPA-OS unterstützt die meisten experimentellen FIPA-„Standards". FIPA-OS wird als „Managed-Open-Source"-Projekt kontinuierlich weiter entwickelt.

FIPA-OS ist in JAVA implementiert und in zwei Versionen erhältlich:

- Standard-FIPA-OS
- Micro-FIPA-OS.

Micro FIPA-OS, entwickelt vor allem an der Universität von Helsinki, zielt dabei auf den Einsatz auf PDA und *Hand-held* Geräten ab.

Die Architektur von FIPA-OS besteht aus

- *mandatory components*,
- *switchable implementation components* und
- *optional components*.

Mandatory Components werden von allen FIPA-OS-Agenten zur Ausführung benötigt, *Switchable Implementation Components* werden zwar benötigt, ihre tatsächliche Implementierung ist aber anwendungsabhängig, und *Optional Components* sind optional und dienen z. B. der Steuerung des Systems.

Das Zusammenspiel der Komponenten wird in der Abbildung 5 erläutert.

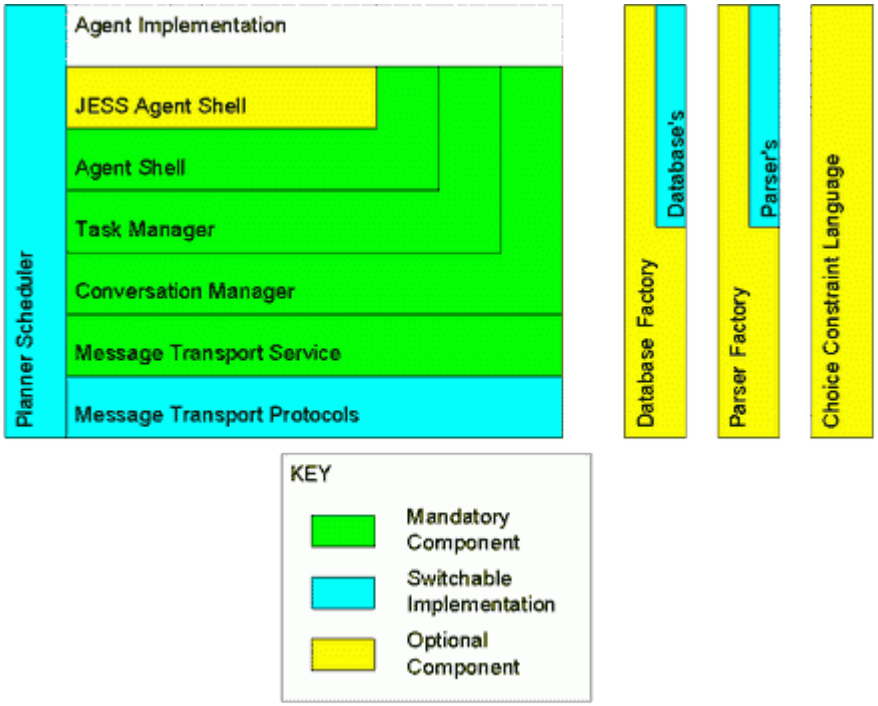


Abbildung 5 FIPA-OS-Architektur [35]

Die *Database Factory*, *Parser Factory* und die *Choice-Constraint-Language*-Komponente sind optional und nicht direkt mit den restlichen Komponenten verzahnt, während der *Planner Scheduler* mit allen anderen Komponenten interagieren kann.

Die *Agent Shell* ist eine abstrakte Komponente, die FIPAOSAgent-Klasse ist eine abstrakte Klasse. Beide dienen jeweils als Basis (-klasse), von der Unterklassen abgeleitet werden, die die eigentliche Agentenimplementierung beinhalten.

Der *Task Manager* hat die Aufgabe, die Funktionalität des Agenten in kleinere, unabhängige Arbeitsgänge, so genannte *Tasks* zu unterteilen. *Tasks* sind abgeschlossene Programmstücke, die eine einzelne spezielle Aufgabe erfüllen und gegebenenfalls die Resultate ihrer Arbeit an den „Auftraggeber“ (ein komplexerer Task) zurückliefern. Tasks haben dabei die Möglichkeit, Nachrichten nach außen zu senden und von außen zu empfangen.

Der *Conversation Manager* regelt und kontrolliert das Versenden und Empfangen von Nachrichten auf dem Level von *Performatives*. Wenn für eine Kommunikation ein bestimmtes Protokoll festgelegt wurde, sei es ein FIPA-Protokoll oder ein anwendungsspezifisches, achtet der *Conversation Manager* auf die Einhaltung des Protokolls.

Die eigentliche Kommunikation wird über den *Message Transport Service* abgewickelt. Der *Message Transport Service* greift dabei wiederum auf ein *Message Transport Protocol* zurück. Das *Message Transport Protocol* ist eine *Switchable-Implementation*-Komponente, so dass verschiedenste Protokolle unterstützt werden können.

In Abbildung 5 ist darüber hinaus auch die optionale Komponente *JESS Agent Shell* abgebildet. Die *JESS Agent Shell* ist eine Erweiterung der Standard *Agent Shell* für das JESS-System [44].

FIPA-OS unterstützt folgende FIPA-Spezifikationen [45]:

- XC00061D FIPA ACL Message Structure Specification

- XC00025D FIPA Interaction Protocol Library Specification
- XC00026E FIPA Request Interaction Protocol Specification
- XC00027E FIPA Query Interaction Protocol Specification
- XC00028E FIPA Request When Interaction Protocol Specification
- XC00029E FIPA Contract Net Interaction Protocol Specification
- XC00030E FIPA Iterated Contract Net Interaction Protocol Specification
- XC00031E FIPA English Auction Interaction Protocol Specification
- XC00032E FIPA Dutch Auction Interaction Protocol Specification.

2.1.5 Zusammenfassung

In diesem Kapitel wurden verschiedene wichtige Aspekte intelligenter Software-Agenten beschrieben.

In einer allgemeinen Einführung in die Theorie der intelligenten Software-Agenten wurden zunächst die Aufgaben und Möglichkeiten intelligenter Software-Agenten im Kontext der interaktiven Simulation und Visualisierung in der computergestützten Biochemie eingeordnet. Anschließend folgte die Beschreibung grundlegender Eigenschaften intelligenter Software-Agenten, wie Autonomie, Reaktivität, Proaktivität, Schlussfolgerungsfähigkeit, Kommunikationsfähigkeit und Persönlichkeit. Im Anschluss hieran wurden grundlegende Agentenarchitekturen vorgestellt.

Nach dieser allgemeinen Einführung lag ein besonderer Schwerpunkt auf der Kommunikation zwischen Agenten in Multi-Agenten-Systemen. Diese sind im Kontext der gekoppelten Simulatoren, die später im konzeptionellen Teil der Arbeit vorgestellt werden, von besonderer Bedeutung. Es wurde deutlich, dass Agentenkommunikationssprachen geeignet sind, den Datenaustausch zwischen verschiedenen Agenten zu ermöglichen. Dabei zeigte sich, dass die Spezifikationen von *Performatives* der FIPA Vorteile gegenüber der KQML-Spezifikation haben. Die FIPA-Performatives werden entsprechend im Kapitel zum Konzept wieder aufgenommen. Das Kapitel beschreibt im Abschluss beispielhaft zwei konkrete Agentenimplementierungen, die die vorher dargelegten Grundlagen erläutern.

2.2. Simulation

Im folgenden Kapitel soll ein Überblick über die Technik der Simulation als Gegenstand und als Mittel der Forschung gegeben werden.

Der Begriff **Simulation** wird nach der VDI-Richtlinie 3633 [46] folgendermaßen definiert:

> "Simulation ist das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem experimentierfähigen Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind."

Nach [47] wird ein **System** wie folgt beschrieben:

> „Ein System ist eine Gesamtheit von Elementen, die willkürlich von Elementen der Umwelt abgetrennt werden. Die Elemente des Systems stehen miteinander und mit den Elementen der Umwelt in Wirkungsbeziehungen, die in Form gerichteter Wirkungsgrößen, den sogenannten Variablen, zu beschreiben sind.“

Dabei ist der Begriff des **Modells** unterschiedlich definiert. Für diese Arbeit wird folgende Definition eines **Simulationsmodells** verwendet:

> Ein Simulationsmodell ist eine Abbildung eines realen Systems, bei [48] „Modelloriginal“, in ein formales Zeichensystem. Zwischen Modell und Original besteht eine *Analogierelation* bezüglich des Modellierungszwecks. Das bedeutet, dass für den Modellierungszweck *relevante* Eigenschaften des Originals analog gelten. Andere, nicht relevante Eigenschaften des Originals werden nicht modelliert, so dass das Modell in der Regel das Original *vereinfacht* abbildet.

Es soll hier bemerkt werden, dass der Modellierungszweck in vielen Fällen *nicht* explizit vom Modellierer formuliert wird, sondern oft stillschweigend Konsens zwischen dem Modellierer und dem Modelladressat ist. Der Modelladressat ist der Benutzer des Modells, der mit dem Modellierer identisch sein kann.

2.2.1 Einsatz von Simulation

Der tatsächliche Einsatz der Simulation in einer Simulationsstudie nimmt, zumindest was den Aufwand angeht, nur einen kleinen Teil der zu veranschlagenden Zeit in Anspruch. In der Praxis hat sich für Simulationsstudien folgender Ablauf etabliert [3] (siehe):

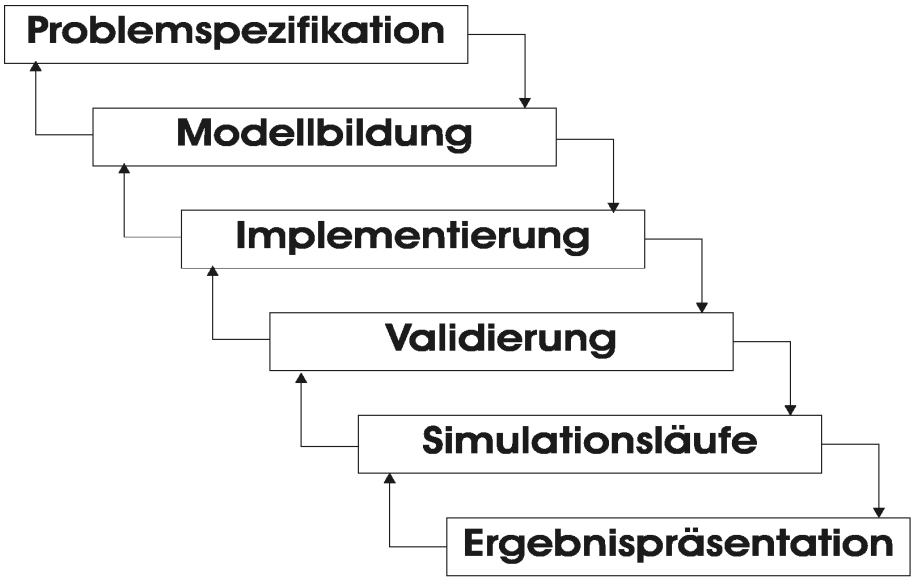


Abbildung 6 Ablauf von Simulationsstudien (nach [3])

- **Problemspezifikation**: Hier wird das zu betrachtende System (siehe oben) umrissen. Dabei wird betrachtet:
  - Welche Elemente gehören zum System, welche gehören zur Umwelt?
  - In welcher Beziehung stehen die Elemente des Systems zueinander?
  - Welche Stellgrößen bzw. welche Schnittstelle zur Umwelt hat das System?
  - Welche Wirkungsgrößen des Systems sind bekannt?

- Welche Parameter des Systems sind bekannt?
- Welche Parameter und Wirkungsgrößen sind quantitativ bekannt bzw. können beobachtet werden?

Sind diese Erkenntnisse gewonnen, so ist zu definieren, in welchen Grenzen die Simulation zum Realsystem in Analogie steht. Dies ist der so genannte *experimentelle Rahmen* der Simulationsstudie. Dieser Ausschnitt des Realsystems hängt ganz entscheidend davon ab, was das Ziel der Simulation und besonders des Modells ist.

Das so genannte Aussageziel, also die Fragestellung, die die Simulationsstudie beantworten soll, lässt sich in zwei Kategorien einteilen: Es werden Probleme, die deskriptive Aussagen benötigen, von Problemen, die präskriptive Aussagen benötigen, unterschieden [49]. Deskriptive Aussagen sind Aussagen der Art: „Das System befindet sich in n Zeiteinheiten im Zustand x“, oder „Das System konvergiert zu einem Gleichgewichtszustand“. Präskriptiv ist etwa folgende Aussage: "Um einen stabilen Zustand x zu erreichen, muss Parameter $p=y$ gewählt werden“.

- **Modellbildung**: Die Modellbildung ist ein zentrales Element der Simulationsstudie. An dieser Stelle muss das System so weit analysiert werden, dass es möglich wird, das Modelloriginal in dem Modell abzubilden. Die Abbildung erfolgt gemäß der oben genannten Definition in ein *formales Zeichensystem*. In vielen Fällen ist dies eine mathematische Beschreibung, z. B. als System von Differenzialgleichungen, als Zustandsübergangsdiagramm eines Automaten, als Regeln eines Regelsystems, als logischer Ausdruck etc. (Dieser Modellbegriff steht im Gegensatz zu beispielsweise physikalischen Modellen, wie etwa Modellflugzeuge etc.)

  Die Tätigkeit des Modellierens ist nur zu einem geringen Teil automatisierbar und bleibt größtenteils Hand- bzw. Kopfarbeit. Dem Modellierer obliegt es hierbei, eine Abbildung zu finden, die das reale System so einfach wie möglich und so detailliert wie nötig wiedergibt. In der Regel ist nach der initialen Modellierung ein weiterer Schritt notwendig: die Modellvereinfachung und Modelltransformation. Ziel dieses Schrittes ist es, die Komplexität des Modells in Hinblick auf die spätere Simulation zu verringern. Diese Vereinfachung und Transformation ist also nicht nur vom Ausgangsmodell, sondern auch von den Eigenschaften der später zu benutzenden Simulationstechnik abhängig. Wie bei der ersten Modellierung ist hier eine Abwägung zu treffen: Zum einen besteht die Forderung, ein in möglichst umfassender Form analoges Modell zu erzeugen. Andererseits gibt es die Notwendigkeit, ein auch numerisch „handhabbares“ Modell zu finden. An dieser Stelle ist der vorher festgelegte Modellzweck von besonderer Bedeutung, denn hierdurch wird der Rahmen der gegebenenfalls nicht auf das Modell zu übertragenden Aspekte des realen Modells abgesteckt.

- **Implementierung**: Die Implementierung, die Umsetzung der Simulationsalgorithmen in ein Computerprogramm, ist aus Sicht der Informatik ein entscheidender Teil einer Simulationsstudie. Anders als für die Modellierung ist es für die Simulationstechnik möglich, generische, für mehrere Anwendungsfälle nutzbare Applikationen zu erstellen. Dies soll nicht davon ablenken, dass es keine einzelne, für alle Anwendungen und alle Studien geeignete Simulationstechnik gibt. Es gibt vielmehr eine direkte Abhängigkeit zwischen dem gewählten Modellierungsverfahren und der Simulationstechnik. Eine Klassifikation von Simulationstechniken folgt im nächsten Unterkapitel. Obwohl die Benennung „Implementierung“ eine *Neu*implementierung suggeriert, kann Implementierung auch die Auswahl eines geeigneten existierenden Simulationssystems sein. Der beschriebene Zusammenhang zwischen

Modellierungstechnik und Simulationstechnik erfordert jedoch eine kritische Betrachtung des „Innenlebens“, also eine Betrachtung der implementierten Verfahren des verfügbaren Systems.

Modelle haben die Gemeinsamkeit, dass sie zwar das System beschreiben, etwa durch Regeln oder Zustandsgleichungen, aber keine Vorgabe machen, *wie* die Regeln zu interpretieren oder die Gleichungen zu lösen sind. Insbesondere für durch mathematische Gleichung beschriebene Modelle gilt, dass es, von simplen, analytisch berechenbaren Modellen abgesehen, in der Regel nicht möglich ist das Modell *exakt* zu berechnen. In den meisten Fällen finden hier am Ende die fehlerbehafteten Methoden der numerischen Mathematik Verwendung. Neben Fehlern bei der Abstraktion in der Modellbildung gibt es also eine weitere mögliche Quelle von Ungenauigkeit und Unsicherheit.

- **Validierung**: Im Schritt der Validierung muss nun die Rechtfertigung der Annahme, dass Modell und Simulationsimplementierung geeignet sind, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind“, überprüft werden (siehe Definition „Simulation“ oben). Die Forderung nach korrekter Programmierung erscheint zwar trivial, ist aber angesichts der Unsicherheit bezüglich der grundsätzlichen Validität des Modells und der Simulationstechnik nicht einfach überprüfbar.

  Unter der Annahme der korrekten Programmierung ist in der Folge die Konsistenz des Modells zu überprüfen. Absurde oder widersprüchliche Ergebnisse sind dabei allerdings unter Umständen auf eine falsche Parameterwahl zurückzuführen. Zu wählende Parameter sind dabei sowohl für Stellgrößen des Modells als auch für Größen der Simulationstechnik, wie etwa Schrittweiten etc., zu ermitteln.

  Zuletzt ist die Korrektheit der Modellannahme zu testen, also die Übereinstimmung der Simulationsergebnisse mit experimentell ermittelten Daten. Hierzu wird in der Regel ein repräsentativer Datensatz ermittelt, der den experimentellen Rahmen beschreibt. Die Entscheidung zur Korrektheit der Modellannahme ist im Licht des Simulationszwecks zu treffen.

  Neben der Korrektheit des Simulationssystems muss auch dessen Angemessenheit überprüft werden. Hauptpunkt ist hier die Berechnungsgeschwindigkeit der Simulation; dies gilt insbesondere, wenn Interaktionsanforderungen an die Simulationsstudie gestellt werden.

- **Simulationsläufe**: Nachdem das Problem beschrieben, ein Modell erstellt, eine Simulationstechnik ausgewählt und implementiert wurde, muss nun die eigentliche Studie durchgeführt werden. Die Durchführung eines Simulationslaufs wird mit der Durchführung eines konventionellen Experiments verglichen [3]. Allerdings wird das Experiment nicht am realen System, sondern am Simulationsmodell durchgeführt.

  Bei der Durchführung eines Simulationslaufs ist über wesentliche Parameter zu entscheiden:

  1. den Anfangszustand des Simulationslaufs
  2. die Länge (Dauer) eines Laufs.

  Beide Entscheidungen sind eng mit der Zielstellung der Studie verbunden. Für die Dauer eines Simulationslaufs unterscheidet man zwei verschiedene *Zeithorizonte*: terminierende und nicht terminierende Probleme. Terminierende Problemfälle haben ein bestimmtes, aus der Aufgabenstellung heraus definierbares Ende, etwa „Simulation

einer Zelle von der Zellteilung des Vorgängers bis zur eigenen Zellteilung“. Die Frage, ob solche Systeme langfristig stationär sind, also in einem Gleichgewicht verharren, ist in der Regel nicht relevant. Anders verhält es sich bei nicht terminierenden Systemen; hier ist der Zeithorizont nicht begrenzt. Ist gleichzeitig das Verhalten der Eingangsgrößen des Systems stabil, so sind Fragestellungen nach einem Gleichgewicht die am häufigsten gestellten.

Für beide Fälle ist die Wahl des Anfangszustands von großer Bedeutung, denn in beiden Fällen können falsch gewählte Anfangswerte zu fehlerhaften Aussagen führen. Bei nicht terminierenden Systemen kann ein falscher Anfangszustand zu im realen System nicht vorhandenen Konvergenzeigenschaften führen. Zudem muss selbst bei korrekter Konvergenz mit einem Einschwingen des Systems gerechnet werden. Für terminierende Systeme können falsche Startbedingungen zu grundsätzlich irrelevanten Simulationsläufen führen.

- **Ergebnisvisualisierung:** Der letzte Schritt einer Simulationsstudie ist die Darstellung des Simulationsergebnisses. Zur Darstellung des Ergebnisses gehören dabei die Dokumentation des Modells mit den zugrunde liegenden Annahmen, die Wahl der Parameter, die Wahl der Simulationstechnik, Ergebnisse der Validierung und die Wahl der Problemlösungsstrategie. All diese Ergebnisse sind notwendig zur Dokumentation und zur korrekten Einschätzung des eigentlichen Ergebnisses der Simulationsstudie. Da diese Ergebnisse ja Experimentergebnissen ähneln (der Unterschied liegt in der Untersuchung des Simulationsmodells anstatt des realen Systems und der Tatsache, dass im Modell unter Umständen Zugriff auf Systemzustände besteht, die der Experimentator beim realen System nicht hat), kommen hier die gleichen Verfahren der Visualisierung zum Einsatz, wie sie aus anderen wissenschaftlichen Visualisierungen bekannt sind. Die Grundlagen der Visualisierung werden in Kapitel 2.3 genauer betrachtet.

2.2.2 Ereignisdiskrete Simulation

Eine der wichtigsten Unterscheidungen in der Simulationstechnik und der Modellierung ist die Unterscheidung in *ereignisdiskrete* und *zeitkontinuierliche* Simulation [49]. Beide Begriffe beschreiben die grundsätzliche Herangehensweise bei der Modellierung von *Zeit* in der Simulation. Die Rolle der Zeit in der Simulation wurde bereits oben unter dem Aspekt des *Zeithorizonts* der Simulation kurz beleuchtet. Nach der Definition der Simulation [46] ist die Betrachtung der *dynamischen Prozesse* eines Systems Inhalt der Simulation. Dynamische Prozesse zeichnen sich durch ihre Veränderlichkeit über die Zeit aus. Damit ist die Zeit als Eingangsparameter sozusagen „gesetzt“.

Die ereignisdiskrete Modellierungssicht geht dabei von der Modellannahme aus, dass der Zustand des betrachteten Systems *nicht* stetigen Veränderungen unterworfen ist, sondern sich vielmehr in Sprüngen ändert. Veränderung in Sprüngen bedeutet, dass sich der Zustand nur an diskreten Zeitpunkten ändert und zwischen diesen Zeitpunkten konstant bleibt. Darüber hinaus wird in vielen Fällen auch der Wertebereich der einzelnen Zustandsvariablen diskretisiert.

Zum Verständnis der ereignisdiskreten Simulation sind einige wichtige Begriffe dieser Kategorie zu betrachten. Im Mittelpunkt stehen hierbei die „Ereignisse“. Ein *Ereignis* beschreibt den Zeitpunkt und den Auslöser eines Zustandswechsels. Das Ereignis selbst hat keine Dauer, sondern nur einen Zeitpunkt.

Zustandswechsel bedeutet hier, dass sich der Wert einer Zustandsvariablen, eines *Attributes* verändert. Ein Attribut ist dabei eine Variable, die den Zustand einer *Entität* beschreibt. Eine Entität ist eine nicht mehr weiter zerlegbare „Grundeinheit“ des Systems. Oft werden in realen

Systemen vorkommende Objekte, wie etwa einzelne Maschinen in einer Produktionssimulation, durch Entitäten modelliert. Eine Entität kann mehrere Attribute haben. Die Werte aller Attribute einer Entität bestimmen den Zustand der Entität. Die Menge der Zustände aller Entitäten eines Systems bestimmt den Gesamtzustand des Systems. Der konstante Wert eines Attributes zwischen zwei dieses Attribut beeinflussenden Ereignissen nennt man *Aktivität*. Weiterhin wird oft zwischen *permanenten* und *temporären Entitäten* unterschieden. Permanente Entitäten, wie etwa die oben angesprochene Maschine, verbleiben während des gesamten Simulationslaufs in der Simulation, während eine temporäre Entität mit einem Ereignis in die Simulation eingeführt oder aus ihr herausgeführt wird. Ein typisches Beispiel für temporäre Entitäten sind so genannte *Jobs*, also Arbeitsaufträge, die in das System gegeben werden und bis zu ihrer Abarbeitung im System verbleiben.

Typische Fragestellungen, die mit Hilfe der ereignisdiskreten Simulation beantwortet werden sollen, sind etwa:

- Fragen nach der Kapazität eines Systems: Wie viele Maschinen, Mitarbeiter, Prozessoren etc. sind notwendig, um einen vorgegebenen Durchsatz zu erreichen?
- Fragen nach der optimalen Auslastung des Systems: Nach welcher Strategie wird eine möglichst hohe Auslastung erreicht?
- Fragen nach der Störanfälligkeit des Systems: Was passiert, wenn eine Maschine ausfällt?
- Logistische Fragen: Wie hoch müssen Lagebestände sein? Wie muss der Materialfluss ausgelegt werden?

2.2.2.1 Beispiel für ein ereignisdiskretes Simulationsmodell

Das folgende einfache Beispiel soll die Idee der ereignisdiskreten Simulation illustrieren. Dabei werden auch zwei wiederkehrende Phänomene beschrieben, die zwar streng genommen nicht notwendig für ein ereignisdiskretes Simulationsmodell sind, aber dennoch typischerweise in solchen Systemen auftauchen: die Modellierung von *Zufall* (stochastische Simulation) und die *Warteschlange*.

Simuliert werden soll die Dynamik eines Ladengeschäftes. In das Geschäft kommen Kunden, die von Verkäufern bedient werden und danach das Geschäft wieder verlassen. Gefragt ist nach der Kapazität des Systems und der mittleren Wartezeit des Kunden bis zu seiner Bedienung.

Als Entitäten werden in diesem Fall identifiziert:

- Kundenpopulation, also die Menge potenzieller Kunden
- Kunden
- Verkäufer, in Abbildung 7 Zentrale Elemente des Warteschlangen-„Bedienstationen“ genannt.

Hinzu kommt noch die Warteschlange, die durch ihr Attribut „Kapazität“ bestimmt, wie viele Kunden sich im Ladengeschäft aufhalten können.

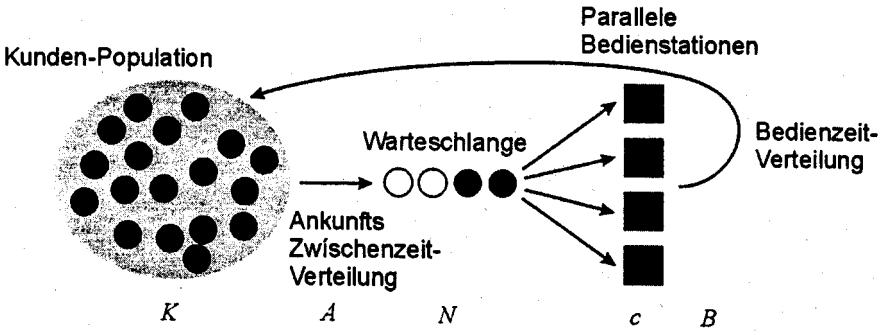


Abbildung 7 Zentrale Elemente des Warteschlangenmodells [3]

Da die Zeitpunkte, an denen neue Kunden von außen in den Laden eintreten, von Faktoren abhängen, die jenseits der Modellierung dieses Modells liegen, d. h. deren genauen Parameter nicht determinierbar sind, wird die Zeit zwischen den Ankünften durch eine Zufallsvariable beschrieben, deren Verteilung ein Attribut der Kundenpopulation ist. Zudem steht die Bearbeitungszeit, die für die Bedienung eines Kunden gerechnet werden muss, nicht fest, sondern wird ebenfalls durch eine Zufallsverteilung beschrieben. (In der Warteschlangentheorie (siehe [50]) wird dabei die gleiche Verteilung für alle Verkäufer angenommen.) Der Einsatz von Zufall in der Simulation wird in einem späteren Kapitel noch ausführlicher beschrieben. In diesem Modell gibt es drei verschiedene Ereignisse:

- Ankunft eines neuen Kunden
- Beginn der Bedienung eines Kunden
- Ende der Bedienung eines Kunden.

Attribut eines Kunden ist z. B. „wartet“ oder „wird bedient“; ein Verkäufer hat etwa das Attribut „bedient“ oder „wartet auf Kunden“.

2.2.2.2 Zeitdiskretisierung

Ein wichtiges Merkmal, das gemeinhin mit ereignisdiskreten Simulationen in Zusammenhang genannt wird, ist die Diskretisierung der Zeitachse. Wie später zu zeigen ist, wird auch bei der kontinuierlichen Simulation *de facto* die Zeitachse diskretisiert. Welche Möglichkeiten gibt es also, diese Diskretisierung durchzuführen? Die Literatur [49] kennt hierzu zwei Konzepte:

- äquidistante Zeitdiskretisierung
- nicht äquidistante Zeitdiskretisierung.

Bei der äquidistanten Zeitdiskretisierung wird die Simulationszeit in gleich großen Sprüngen vorwärts geschrieben (konstantes Δt), während bei nicht äquidistanter Zeitfortschreibung die Sprünge entsprechend nicht gleich groß sind. Für die ereignisdiskrete Simulation bietet es sich an, die Diskretisierung mit einem Ereignis zusammenfallen zu lassen (siehe Abbildung 8 Zeitdiskretisierung). Bei dieser Variante wird das Simulationsmodell genau beim Eintritt eines Ereignisses betrachtet, also genau dann, wenn nach dem Paradigma der ereignisdiskreten Simulation ein Zustandswechsel eintreten kann.

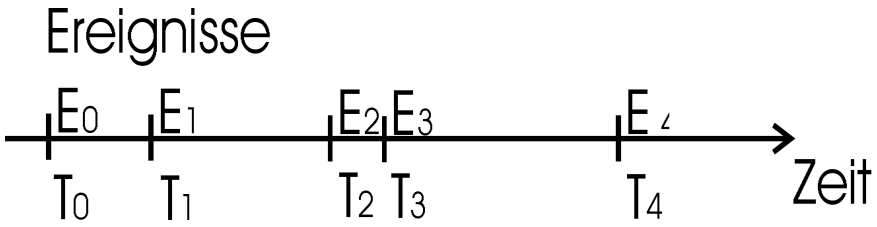


Abbildung 8 Zeitdiskretisierung

Die andere Variante der Zeitfortschreibung ist die der äquidistanten Zeitintervalle. Obwohl diese Variante zunächst einfacher erscheint, hat sie für die ereignisdiskrete Simulation einige Nachteile (siehe Abbildung 9 Äquidistante Zeitdiskretisierung)

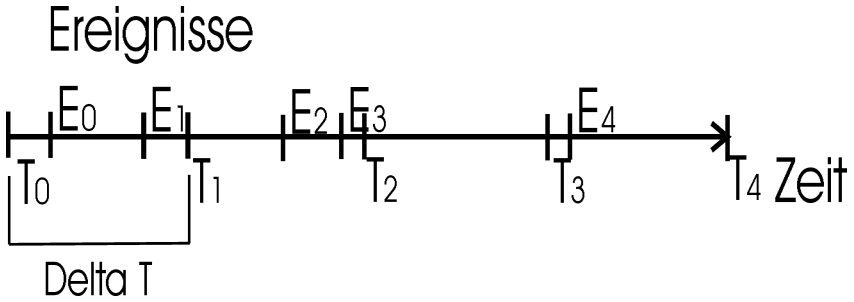


Abbildung 9 Äquidistante Zeitdiskretisierung

Der erste Nachteil ist, dass für Ereignisse, die zwischen zwei Zeitsprüngen auftreten, keine zeitliche Auflösung mehr möglich ist. Dies bedeutet, dass dieser Konflikt künstlich, etwa durch Regeln, aufgelöst werden muss. Der zweite Nachteil ist, dass für äquidistante Zeitinkremente jeweils das Eintreffen neuer Ereignisse zu berechnen ist. Je nach Verteilung der Zufallsvariablen, die das Eintreffen eines weiteren Ereignisses steuert, ist diese Berechnung der bedingten Wahrscheinlichkeit *P (Eintreffen eines Ereignisses im nächsten Zeitintervall | Es gab bisher kein Ereignis)* sehr schwierig. Ist bereits bekannt, etwa durch Vorausberechnung, dass im folgenden Intervall kein Ereignis stattfinden wird, ist es unnötig, für diesen Zeitsprung Berechnungen durchzuführen, so dass wieder der oben beschriebene Fall der durch das Auftreten eines Ereignisses gesteuerte, nicht äquidistante Fall der Zeitfortschreibung eintritt.

2.2.2.3 Methoden der ereignisdiskreten Simulation

Die Grundvorgehensweise der ereignisdiskreten Simulation geht nicht von einer äquidistanten, sondern von der oben beschriebenen nicht äquidistanten Zeitfortschreibung aus.

Bei dieser Vorgehensweise wird die Simulationszeit initialisiert, und die Simulation beginnt. Je nach Modell wird, meist bestimmt durch die Ausprägung einer Zufallsvariablen, eine Reihe von Ereignissen erzeugt. Einem Ereignis ist stets, wie oben bereits beschrieben, ein Zeitpunkt zugeordnet. Die erzeugten Ereignisse werden entsprechend ihren Zeitpunkten in einer Liste geordnet. Das erste Element in dieser Liste ist somit das Element, das dem Startzeitpunkt am nächsten liegt. Mithin ist offensichtlich, dass zwischen dem Start und diesem ersten Ereignis keine Veränderungen im System erfolgen, denn Zustandsänderungen sind nur bei Eintreffen eines Ereignisses zulässig. Die Simulationszeit wird also bis zum Zeitpunkt des ersten Ereignisses vorgestellt, das Ereignis wird in das System eingebracht, Zustandsänderungen werden ausgelöst. In der Regel werden hierbei neue Ereignisse generiert, die wiederum in die

Ereignisliste einsortiert werden, die Zeit wird zum nächsten Ereignis weitergestellt etc. Die Ereignisliste ist die Grundstruktur der ereignisdiskreten Simulation.

Zusammenfassend kann der Grundalgorithmus (in einer Pseudoprogrammiersprache) also folgendermaßen beschrieben werden:

```
InitializeSystem();
EventList = GenerateInitialEvents();
SortEventList(EventList);
While(EventList.hasMoreElements())
{
      event = EventList.getFirstElement();
      EventList.removeFirstElement();
      NewEvents = EventRoutine(event);
      SortEventList(EventList);
}
AnalyzeSimulationResults();
```

Die wichtigsten Funktionen sollen hier noch einmal betrachtet werden.

`InitializeSystem()`: In dieser Funktion wird das System initialisiert. Diese Aufgabe ist nur scheinbar trivial; wie bereits oben beschrieben wurde, ist die Auswahl der Startwerte und Parameter von entscheidender Bedeutung für die erreichbaren Ergebnisse der Simulationsstudie.

`GenerateInitialEvents()`: Die Simulation läuft, falls nicht ein dedizierter Zeithorizont für die Simulation vorgegeben wird, solange sich Ereignisse in der Ereignisliste befinden. Ohne initiale Events wird die Simulation nicht gestartet.

`SortEventList(EventList)`: Die Ereignisse werden nach Zeitstempel geordnet.

`EventRoutine(event)`: Die Event Routine ist das Herzstück der Simulation. In dieser Routine wird das Ereignis ausgewertet. Auswertung bedeutet, dass

1. die Zustandswechsel durchgeführt werden und
2. (statistische) Daten erhoben werden, die zur Auswertung der Simulationsstudie notwendig sind.

`AnalyzeSimulationResults()`: Am Ende der Simulation sind die Ergebnisse der Studie zusammenzustellen, zu berechnen und zu visualisieren.

Ergebnisvisualisierung findet nicht notwendigerweise am Ende der Studie statt, je nach Anforderung ist dies auch während des Simulationslaufs möglich.

2.2.2.4 Die Rolle des Zufalls in der ereignisdiskreten Simulation

In vorigen Kapiteln wurde bereits darauf hingewiesen, dass die Modellierung des Zufalls eine besondere Rolle der ereignisdiskreten Simulation darstellt. Dies geschah allerdings immer im

Zusammenhang der Modellierung von Eingangsgrößen oder Bearbeitungszeiten. Der Einsatz von Zufallsvariablen wird oft dann in Erwägung gezogen, wenn die darzustellenden Prozesse zu komplex sind, um sie geschlossen darzustellen, wenn keine oder zu wenige Informationen über den betrachteten Prozess vorhanden sind oder wenn ein natürlicher, zufälliger Vorgang betrachtet wird (z. B. radioaktiver Zerfall).

Eine wichtige Problemstellung in all diesen Fällen ist in der Regel die Auswahl der Wahrscheinlichkeitsverteilung der Zufallsvariablen. Zu dieser Auswahl sei auf einschlägige Literatur der Stochastik verwiesen. [51].

Aus technischer Sicht birgt dieses Vorgehen das Problem der Erzeugung unabhängiger Zufallszahlen, die für die Erzeugung von konkreten Stichproben der vorher bestimmten Verteilung benötigt werden. Das Thema der Konstruktion von Zufallszahlengeneratoren verlässt den Rahmen dieser Arbeit, daher sei auch hier auf die einschlägige Literatur verwiesen [52].

2.2.2.5 Monte-Carlo-Simulation

Über die eben beschriebenen Einsatzgebiete des Zufalls hinaus hat sich eine eigene, durch den Einsatz des Zufalls charakterisierte Simulationstechnik entwickelt, die so genannte *Monte-Carlo-Simulation*. Die Monte-Carlo-Simulation wurde erstmalig im Zusammenhang mit der Entwicklung der Atombombe in den 1940er Jahren beschrieben und erhielt auch hier ihren Namen. Die Grundidee dieser Familie von Techniken liegt darin, dass analytisch oder sogar numerisch schwer berechenbare Größen mit Hilfe von Zufallsexperimenten geschätzt werden. Für die technische Umsetzung derartiger Verfahren sind wiederum die im vorigen Abschnitt angesprochenen Zufallszahlengeneratoren von Bedeutung.

Im Folgenden sollen zwei einfache Methoden der Monte-Carlo-Simulation betrachtet werden, das so genannte *Hit-and-Miss*- und das *Crude*-Monte-Carlo-Verfahren [49]. Beide Verfahren werden verwendet, um deterministische Probleme mit Hilfe von Zufallsexperimenten zu lösen.

Das *Hit-and-Miss*-Monte-Carlo-Verfahren lässt sich am besten durch ein einfaches Beispiel illustrieren [49]:

Problemstellung: Der Flächeninhalt einer Fläche ist zu berechnen. Die Fläche sei gegeben durch ein Integral der folgenden Form:

$$A = \int_0^1 f(x)dx$$

Gleichung 2 Flächeninhalt unter einer Kurve

Als Beispiel sei hier *f(x) = x* gesetzt. Die gesuchte Fläche ist die Fläche unterhalb der ersten Winkelhalbierenden. Statt nun das Integral analytisch zu lösen (was bei diesem einfachen Beispiel problemlos möglich wäre), wird die Fläche durch folgendes Zufallsexperiment ermittelt:

Es werden zwei Folgen, x_i und y_i, von U(0,1) verteilten Zufallszahlen erzeugt. Die U(0,1)-Verteilung bedeutet eine Gleichverteilung (auch Rechteckverteilung genannt) im Intervall [0,1]. Interpretiert man die (x_i, y_i) als Punkte in $\Re^2$, so decken diese die Fläche des Einheitsquadrats gleichmäßig ab. Abbildung 10 Hit-/ Miss-Monte-Carloverdeutlicht den Zusammenhang.

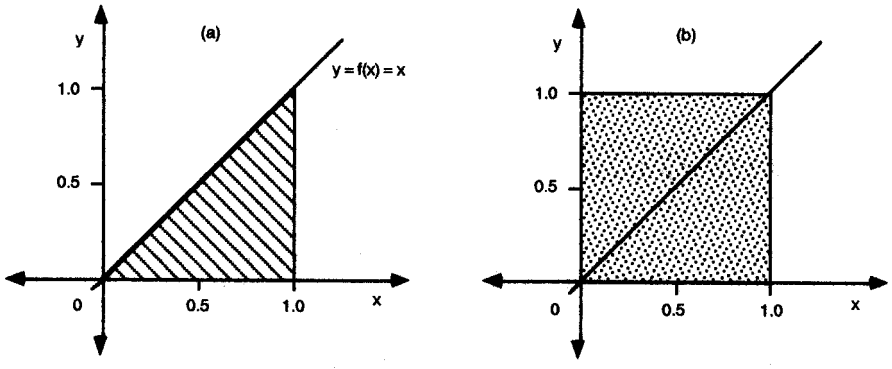


Abbildung 10 Hit-/ Miss-Monte-Carlo [49]

Für jeden Punkt kann nun leicht geprüft werden, ob der Punkt unterhalb (= *Hit*) oder oberhalb (= *Miss*) der gesuchten Funktion liegt. Im Beispiel gilt:

Hit: $y_i \leq f(x_i) = x_i$

Miss: $y_i > f(x_i) = x_i$

Die Fläche unter *f(x)* lässt sich nun durch die Wahrscheinlichkeit P (*Hit*) abschätzen. Nach dem Gesetz der großen Zahl konvergiert die relative Häufigkeit gegen die Wahrscheinlichkeit mit größer werdendem Stichprobenumfang.

Das zweite hier vorgestellte Verfahren, das *Crude*-Monte-Carlo, soll anhand des gleichen Beispiels erklärt werden. Das Grundprinzip dieses Verfahrens macht sich folgenden Zusammenhang zunutze:

Der Erwartungswert *E(X)* einer stetig verteilten Zufallsvariablen *X* mit Dichte *f*(x) ist definiert als:

$$E(X) = \int_{-\infty}^{\infty} x f(x)dx \text{ ; falls } \int_{-\infty}^{\infty} |x| f(x)dx < \infty \text{ gilt.}$$

Gleichung 3 Erwartungswert

Analog gilt:

$$E(g(X)) = \int_{-\infty}^{\infty} g(x) f(x)dx$$

Gleichung 4 Erwartungswert zusammengestzten Funktion

Der Erwartungswert *E(X)* lässt sich durch das arithmetische Mittel einer Stichprobe x_i abschätzen. Das arithmetische Mittel ist ein erwartungstreuer Schätzer für *E(X)*). Es gilt also:

$$\frac{1}{n}\sum_{i=1}^{n} x_i \cong E(x)$$

Gleichung 5 Arithmetisches Mittel als Schätzer für den Erwartungswert

Gelingt es also, ein Problem als Integral der Form

$$\int_{-\infty}^{\infty} g(x) f(x) dx$$

zu formulieren, dann lässt sich der Wert dieses Integrals als

$$E(g(X)) \approx \frac{1}{n}\sum_{i=1}^{n} g(x_i)$$

Gleichung 6 Schätzer der verknüpften Funktion

abschätzen.

Ein beliebiges Integral $\int_a^b h(x)dx$ muss also in eine Funktion *g(x)* und eine „Dichtefunktion“ $f(x) = \frac{1}{b-a}$ faktorisiert werden, um zu der Aussage

$$\int_a^b h(x)dx = \int_a^b g(x) f(x) dx = E(g(x)) \approx \frac{1}{n}\sum_{i=1}^{n} g(x_i)$$

Gleichung 7 Faktorisierung einer Funktion

zu gelangen. Somit nutzt das Verfahren der Monte-Carlo-Simulation Elemente der zeitdiskreten Simulation, um Probleme zu lösen, die gewöhnlich der zeitkontinuierlichen Simulation zugerechnet werden. Die Methoden der zeitkontinuierlichen Simulation werden im Folgenden beschrieben.

2.2.3 Zeitkontinuierliche Simulation

Wird bei der zeitdiskreten Simulation angenommen, dass sich der Zustand des Systems nur an diskreten Zeitpunkten ändern kann, so geht die *zeitkontinuierliche* Simulation von einer kontinuierlichen Änderung aus. Das Herzstück der zeitdiskreten Simulation war die *EventRoutine*. Bei der Beschreibung dieser Routine wurde offen gelassen, *wie* die Auswertung eines Events durchgeführt wird, also nach welchen Regeln oder Funktionen die Zustandsänderungen berechnet werden. Im Gegenzug war der Mechanismus der Generierung und Verwaltung der Events von entscheidender Bedeutung. Bei der zeitkontinuierlichen Simulation gibt es diesen Mechanismus nicht, denn in reinen kontinuierlichen Systemen wird dieser Ansatz nicht benutzt. (Es gibt so genannte *hybride Systeme*, in denen beide Simulationstechniken vereinigt werden. [53]). Die zentralen Elemente zeitkontinuierlicher

Systeme sind die Zeitfortschreibung und die adäquate Modellbeschreibung. Die Modellbeschreibung ist selbstverständlich auch für die zeitdiskrete Simulation wesentlich, aber für die kontinuierliche Simulation gibt es einen engen Zusammenhang zwischen Zeitdiskretisierung und Modellbildung. Dies bedeutet, dass die verwendete Modellierung direkten Einfluss auf die verwendete Technik hat, woraus sich wiederum Anforderungen an die Zeitdiskretisierung ergeben.

Die am häufigsten eingesetzte Modellierungstechnik in der zeitkontinuierlichen Simulation ist die Formulierung in Form von Systemen von Gewöhnlichen Differenzialgleichungen (Gewöhnliche Differenzialgleichung (DGL), Ordinary Differential Equations (ODE)). Die allgemeine Darstellung eines Systems von ODE 1. Ordnung ist gegeben durch

$\dot{x} = f(x, i, p, t)$, wobei

x den Vektor der Zustandsvariablen,

i den Vektor der Inputgrößen,

p den Vektor der Systemparameter und

t die Systemzeit beschreibt.

Zur vollständigen Beschreibung fehlt nun noch x_0, der Vektor der Start- oder Randwerte.

Die direkte Formulierung der Differenzialgleichungen ist oftmals ein unübersichtliches Unterfangen, insbesondere wenn das gesamte Modell nicht in einem Schritt, sondern inkrementell und in verschiedenen Schritten erstellt werden soll. Eine Analogie aus dem Bereich der Softwareentwicklung wäre etwa die Programmierung von „Spaghetti-Code“; das Ergebnis ist unübersichtlich und schlecht wartbar.

Auswege bilden Modularisierungskonzepte, wie sie etwa in Matlab oder SIMULINK [54] oder ASPEN [55] umgesetzt sind. Hierbei wird dem Benutzer eine graphische Benutzungsoberfläche geboten, in der graphische Repräsentationen von *Modulen* in an Schaltkreisen erinnernde Netzpläne angeordnet und verbunden werden können. Diese *graphisch modellierte* Darstellung wird anschließend in Gleichungsform übersetzt.

Gerade für modularisierte Systeme tritt der Fall auf, dass die entstehenden Systemgleichungen nicht nur *reine* Differenzialgleichungen enthalten. Es entsteht vielmehr ein gemischtes System von Differenzial- und algebraischen Gleichungen (Differential-Algebraic Equations, DAE) der Form

$$\dot{x} = f(t, x, y)$$
$$0 = g(t, x, y)$$

Gleichung 8 Differenzial-Algebraisches Gleichungssytem (DAE)

Der Vektor *x* fasst dabei sämtliche Variablen zusammen, die auch in ihrer Ableitung, $\dot{x}$, vorkommen. Der Vektor *y* fasst die restlichen Variablen zusammen. Aus diesem Zusammenhang ergeben sich zusätzliche Anforderungen an die numerische Simulationstechnik.

2.2.3.1 Mehrdimensionale zeitkontinuierliche Systeme

Bisher wurden Systeme betrachtet, deren Variablen allein von der Zeit abhängig waren. Solche Variablen werden auch als *extensive* Variablen bezeichnet, sie haben keine räumliche Veränderlichkeit. Dem gegenüber stehen *intensive* Variablen, die neben der Zeit auch von ihrem Ort im Raum abhängig sind. Typische Beispiele hierfür sind:

- die Dichte eines inhomogenen Körpers
- die Konzentration von gelösten Stoffen
- die Temperaturverteilung innerhalb eines Körpers.

Während bei räumlich homogenen Systemen die eine differenzierte Größe *t*, die Zeit, zu gewöhnlichen Differenzialgleichungssystemen führt, sind nun mehrere Größen zu differenzieren. Dies führt entsprechend zu Partiellen Differenzialgleichungssystemen (PDE). In R^3 sind also vier Größen zu betrachten, die drei Dimensionen des Raums und die Zeit.

2.2.3.2 Methoden der zeitkontinuierlichen Simulation

Die im Folgenden vorgestellten Methoden der zeitkontinuierlichen Simulation sind im Wesentlichen Methoden der numerischen Mathematik zur Lösung von (Differenzial-) Gleichungssystemen. Damit sind sie nicht notwendigerweise originär für die Simulation, und ihre genaue Diskussion geht über den Rahmen dieser Arbeit hinaus. Hierzu sei auf die entsprechende Fachliteratur verwiesen [56]. Im Folgenden sollen die für die Simulation wichtigsten Gruppen von Methoden beschrieben werden. Die Frage der Diskretisierung der Zeit (und später auch des Raums) wird hier noch einmal gesondert betrachtet.

Die Verfahren der Numerik, die sich mit der Lösung gewöhnlicher Differenzialgleichungen beschäftigen, werden *Integratoren* genannt. Diese im ersten Moment eigenwillige Namensgebung rührt von folgendem einfachen Zusammenhang her:

Die Lösung des Integrals

$$x(t) = \int_o^T f(t)dt$$

lässt sich als einfacher Spezialfall einer gewöhnlichen Differentialgleichung formulieren, als

$$\dot{x} = f(t); x(0) = 0$$

Im Folgenden soll mit dem expliziten und dem impliziten Euler-Verfahren und dem Runge-Kutta-Verfahren sowie Überlegungen zu Mehrschrittverfahren ein Überblick über die gängigsten Verfahren gegeben werden.

Obwohl dieses Kapitel mit „Methoden der zeitkontinuierlichen Simulation“ überschrieben ist, lassen sich kontinuierliche Aussagen über den Systemzustand nur treffen, wenn sich die für die Beschreibung des Systems verwendeten Regeln und (Differential-) Gleichungen analytisch lösen lassen. Dies lässt sich im Allgemeinen *nicht* annehmen. Daher müssen zur Lösung der beschriebenen Systeme numerische Methoden herangezogen werden. Diese Methoden zeichnen sich allesamt durch eine diskrete Zeitschrittweite aus. Anders als bei der ereignisdiskreten Simulation wird jedoch nicht die Annahme getroffen, dass der Systemzustand zwischen zwei Zeitschritten konstant bleibt und sich nur am Zeitschritt in einem Sprung ändert. Die Schrittweite Δt ist dabei nicht willkürlich wählbar, vielmehr haben die betrachteten Verfahren verschiedene Eigenschaften in Bezug auf die Schrittweite.

2.2.3.2.1 Explizites Euler-Verfahren

Das Grundprinzip der numerischen Lösung von Differentialgleichungen wird am einfachsten dieser Verfahren, dem *expliziten Euler-Verfahren* deutlich.

Erweitert man den oben beschriebenen Zusammenhang zu

$\dot{x} = f(t,x); x(0) = 0$,

dann wird eine Näherungslösung für die Ableitung $\dot{x}$ gesucht. Diese Ableitung lässt sich durch einen Differenzenquotienten nähern:

$$\dot{x}(i\Delta t) \approx \frac{x((i+1)\Delta t) - x(i\Delta t)}{\Delta t}$$

Gleichung 9 Differenzenquotient

Aus $f(i\Delta t, x(i\Delta t)) = \dot{x}(i\Delta t)$ lässt sich folgende Näherung für $x((i+1)\Delta t)$ formulieren:

$$x((i+1)\Delta t) = x(i\Delta t) + \Delta t \cdot f(i\Delta t, x(i\Delta t))$$

Gleichung 10 Euler explizit

Dies ist der Berechnungsschritt des expliziten Euler-Verfahrens. Das Verfahren heißt *explizit*, da der neue Wert von *x* explizit aus bereits berechneten Werten von *x* berechnet wird.

2.2.3.2.2 Runge-Kutta-Verfahren

Das vorgestellte Euler-Verfahren ist das einfachste der expliziten Verfahren. Leider ist die *Fehlerordnung* des Verfahrens zu schlecht für die praktische Nutzung. Die *Fehlerordnung* eines Verfahrens ist ein Maß für den *globalen Diskretisierungsfehler*, also den Fehler zwischen dem analytisch exakten Wert von *x(t)* und dem numerisch genäherten Wert. Dieser globale Fehler ist die Folge von drei Fehlerquellen:

- Rundungsfehler durch begrenzte Rechengenauigkeit
- lokale Diskretisierungsfehler, also die Differenz zwischen dem genäherten Wert und dem exakten Wert an einem konkreten Zeitpunkt t_k
- Fortpflanzung des lokalen Fehlers.

Der Fehler wird durch den Vergleich des betrachteten Verfahrens mit einer Taylor-Reihenentwicklung der betrachteten Funktion gebildet. Er hat die Größenordnung $O(\Delta t^p)$ mit Δt als Schrittweite des Verfahrens und p als *Fehlerordnung*. Für ein Verfahren der Fehlerordnung *p* gilt, dass sich der globale Diskretisierungsfehler bei einer Halbierung der Schrittweite um den Faktor 2^p verringert. Das angesprochene Euler-Verfahren ist von Ordnung 1.

Die Familie der Runge-Kutta-Verfahren enthält Verfahren höherer Ordnung. Der allgemeine Ansatz ist folgender:

$$x((i+1)\Delta t) = x(i\Delta t) + \Delta t(c_1 f(t_1, \bar{x}_1) + c_2 f(t_2, \bar{x}_2) + \ldots)$$

Gleichung 11 Allgemeines Schema des Runge-Kutta-Verfahrens

Die t_i sind aus dem Intervall $[i\Delta t,(i+1)\Delta t]$ und haben die Form

$t_i = i\Delta t + a_i \Delta t$,

und die entsprechenden $\bar{x}_i$ sind:

$\bar{x}_i = x(i\Delta t) + b_{i,1}\Delta t \cdot f(t_1,\bar{x}_1) + b_{i,2}\Delta t \cdot f(t_2,\bar{x}_2) + \ldots$

Die Parameter $a_i, b_{i,k}, c_i$ sind so zu wählen, dass die gegebene Form möglichst ähnlich der ersten Glieder der Taylorreihenentwicklung wird.

Das „klassische“ Runge-Kutta-Verfahren ist vierstufig und von Ordnung 4 und hat die Koeffizienten:

$$
\begin{aligned}
&a_1 = 0,\ a_2 = \tfrac{1}{2},\ a_3 = \tfrac{1}{2},\ a_4 = 1,\\
&b_{1,1} = 0,\ b_{2,1} = \tfrac{1}{2},\ b_{2,2} = 0,\\
&b_{3,1} = 0,\\
&b_{3,2} = \tfrac{1}{2},\\
&b_{3,3} = 0,\\
&b_{4,1} = 0,\\
&b_{4,2} = 0,\\
&b_{4,3} = \tfrac{1}{2},\\
&b_{4,4} = 0,\\
&c_1 = \tfrac{1}{6},\\
&c_2 = \tfrac{1}{3},\\
&c_3 = \tfrac{1}{3},\\
&c_4 = \tfrac{1}{6}
\end{aligned}
$$

Gleichung 12 Koeffizienten des Runge-Kutta-Verfahrens 4. Ordnung

2.2.3.2.3 Implizites Euler-Verfahren

Einen anderen Weg geht das *implizite Euler-Verfahren*. Das implizite Euler-Verfahren verwendet den gleichen Differenzenquotientenansatz wie das explizite Verfahren. Diesmal wird jedoch die Ableitung nicht für den *i.* Zeitschritt, sondern für den *(i+1).* Zeitschritt angenähert (sieheAbbildung 11 Implizites Eulerverfahren).

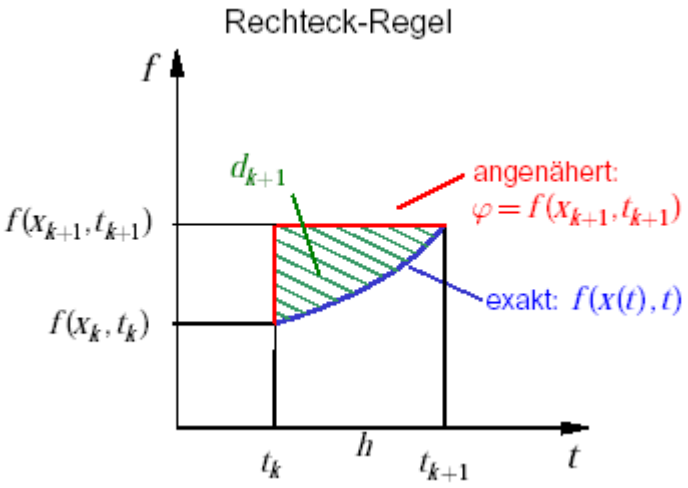


Abbildung 11 Implizites Eulerverfahren [183]

$$f((i+1)\Delta t, x((i+1)\Delta t)) = \dot{x}((i+1)\Delta t) \approx \frac{x((i+1)\Delta t) - x(i\Delta t)}{\Delta t}$$

Gleichung 13 Näherung für implizites Euler Verfahren

Offensichtlich lässt sich diese Näherungsgleichung nur dann nach dem gesuchten $x((i+1)\Delta t)$ auflösen, wenn *f(t,x)* nach *x* auflösbar ist. Für den linearen Fall $f(t,x) = a(t)x + b(t)$ gilt:

$$x((i+1)\Delta t) = \frac{x(i\Delta t) + \Delta t \cdot b((i+1)\Delta t)}{1 - \Delta t \cdot a((i+1)\Delta t)}$$

Gleichung 14 Euler implizit für in *x* lineare *f(t,x)*]

Die hier gezeigte Formel geht von einer eindimensionalen Variable *x* aus, für den mehrdimensionalen Fall wird *b* zu einer Vektorfunktion, und *a* ist eine Matrix (*1* wird zur Einheitsmatrix).

Lässt sich *f(t,x)* nicht explizit nach *x* auflösen, so muss für $x((i+1)\Delta t)$ ein (nichtlineares) Gleichungssystem der Form

$$x = g(x)$$

Gleichung 15 Fixpunktgleichungssystem

gelöst werden. Es gibt eine große Mannigfaltigkeit verschiedener Lösungsansätze für die numerische Lösung von Gleichungssystemen. Für eine Diskussion dieser Verfahren sei auf die Literatur verwiesen [56].

2.2.3.2.4 Stabilität numerischer Integratoren

Ebenso wie für das Euler-Verfahren gibt es auch für Runge-Kutta-Verfahren implizite Verfahren. Wie für das implizite Euler-Verfahren gezeigt, ist in der Regel eine numerische Lösung eines (linearen oder nicht linearen) Gleichungssystems nötig. Dieser aufwändige Schritt ist in expliziten Verfahren nicht nötig. Worin liegt nun der Vorteil der Nutzung impliziter Verfahren?

Die Beantwortung dieser Frage steht im Zusammenhang mit der Frage nach der *Stabilität* des betrachteten numerischen Verfahrens. Betrachtet man zunächst das Ausgangssystem für die Modellierung, so zeigt sich, dass es eine Klasse von Problemen gibt, die für die Betrachtungen in Simulationen von besonderer Bedeutung sind: Dies sind Systeme mit einem Fixpunkt oder *stationären Zustand*. Ein solcher stationärer Zustand ist ein Systemzustand, der, einmal erreicht, trotz Fortschreibung der Zeit nicht wieder verlassen wird. In manchen Systemen sind das etwa Gleichgewichtszustände, Zustände minimaler Energie etc. Von einem *stabilen* numerischen Verfahren spricht man nun, wenn auch die numerische Lösung, wie das betrachtete System, gegen diesen Fixpunkt strebt.

Für das bereits bekannte Anfangswertproblem

$$\dot{x}(t) = f(t, x(t)); x(t_0) = c$$

Gleichung 16 Anfangswertproblem

lassen sich iterative Einschrittverfahren allgemein formulieren als

$$x((i+1)\Delta t) = A(i\Delta t, x(i\Delta t)).$$

Gleichung 17 Iterative Einschrittverfahren allgemein

Für das explizite Euler-Verfahren gilt demnach:

$$A(t, x(t)) = x(t) + \Delta t \cdot f(t, x(t)).$$

Gleichung 18 Euler explizit als Einschrittverfahren

Für den *stationären Zustand* x_∞ gilt demnach:

$$x_\infty = A(t, x_\infty); t \to \infty.$$

Gleichung 19 Fixpunktgleichung für den stationären Zustand

Erfüllt das Verfahren die Lipschitzbedingung [56]

$$\left| \frac{\partial A(t,x)}{\partial x} \right| < 1,$$

Gleichung 20 Lipschitzbedingung

so ist *A(t,x) kontraktiv*, und $x((i+1)\Delta t) = A(i\Delta t, x(i\Delta t))$ konvergiert für

$$\left|\frac{\partial A(t,x)}{\partial x}\right|_{x=x_\infty} < 1$$

Gleichung 21 Stabilitätsbedingung

Betrachtet man beispielsweise wiederum das explizite Euler-Verfahren für *f(t,x(t))* und skalare *x*, erhält man:

$$\frac{\partial A(t,x)}{\partial x} = 1 + \Delta t \frac{\partial f(t,x(t))}{\partial x}$$

Gleichung 22 Stabilität des Euler-Verfahren

Die Stabilitätsbedingungen lauten also:

$$\left|1 + \Delta t \frac{\partial f(t,x(t))}{\partial x}\right| < 1$$

Gleichung 23 Stabilität des expliziten Euler-Verfahrens (1)

Die partielle Ableitung wird für mehrdimensionale *x* (und *f*) zu einer Matrix. Es lässt sich zeigen, dass zur Untersuchung der Stabilität die Eigenwerte λ dieser Matrix anstelle der Ableitung betrachtet werden können. Damit wird aus der Stabilitätsbedingung für das Euler-Verfahren:

$$\left|1 + \Delta t \cdot \lambda\right| < 1$$

Gleichung 24 Stabilität des expliziten Euler Verfahrens (2)

Beachtet man, dass die Eigenwerte komplexwertig sein können, so ergibt sich ein Stabilitätsbereich:

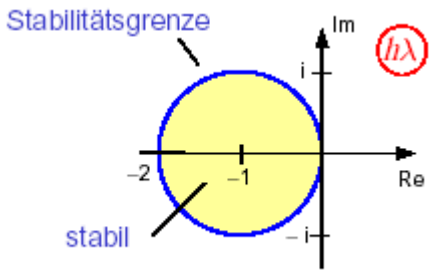


Abbildung 12 Bild Stabilität Euler explizit (Δt wird hier als hλ bezeichnet) [183]

Man sieht, dass zur Einhaltung der Stabilitätsgrenzen $\Delta t \sim \frac{1}{\lambda}$ gilt. Leider sind einige relevante Probleme *steif*, d. h. $\mathrm{Re}(\lambda_{\min}) << \mathrm{Re}(\lambda_{\max})$, und der Realteil des kleinsten Eigenwerts wird betragsmäßig relativ groß, so dass Δt sehr klein werden muss. Das gleiche Problem ergibt sich für alle expliziten Verfahren.

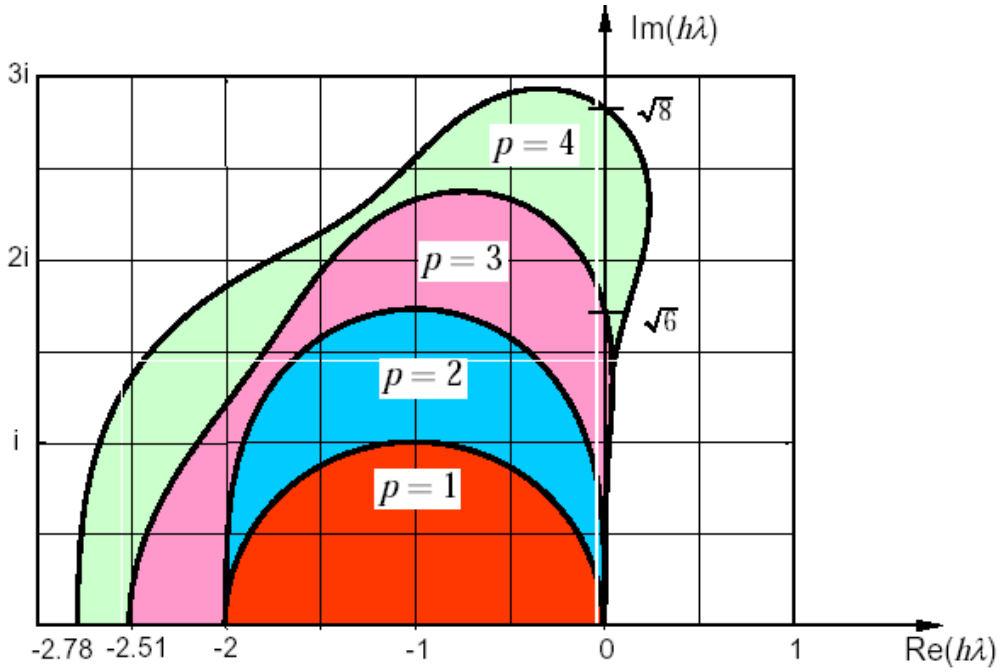


Abbildung 13 Stabilitätsbereich verschiedener Runge-Kutta-Verfahren [183]

Anders ist dies für implizite Verfahren, hier wieder beispielhaft am impliziten Euler-Verfahren beschrieben. Betrachtet man wiederum die Eigenwerte der Ableitungsmatrix, so ergibt sich für das implizite Euler-Verfahren folgende Stabilitätsbedingung (siehe auch Gleichung 13 Näherung für implizites Euler Verfahren):

$$\left| \frac{1}{1 - \Delta t \cdot \lambda} \right| < 1$$

Gleichung 25 Stabilitätsbedingung Euler implizit

Das implizite Euler-Verfahren, ebenso wie die anderen impliziten Verfahren, ist also für alle Schrittweiten Δt stabil.

2.2.3.2.5 Mehrschrittverfahren

Neben den oben beschriebenen Einschrittverfahren werden noch so genannte *Mehrschrittverfahren* verwendet. Es gibt wiederum explizite und implizite Verfahren mit den

beschriebenen grundlegenden Eigenschaften. Das Grundprinzip der Mehrschrittverfahren nutzt bereits berechnete *x(t)* für die Extrapolation neuer Werte für *x(t)*. Ausgegangen wird dabei von der Integralform von $\dot{x}(t) = f(t,x)$:

$$x((i+1)\Delta t) = x(i\Delta t) + \int_{i\Delta t}^{(i+1)\Delta t} f(x(\tau)) \cdot d\tau$$

Die Funktionswerte *f(x(t))* werden aus bekannten Werten *x(t) extrapoliert* (siehe Abbildung 14)

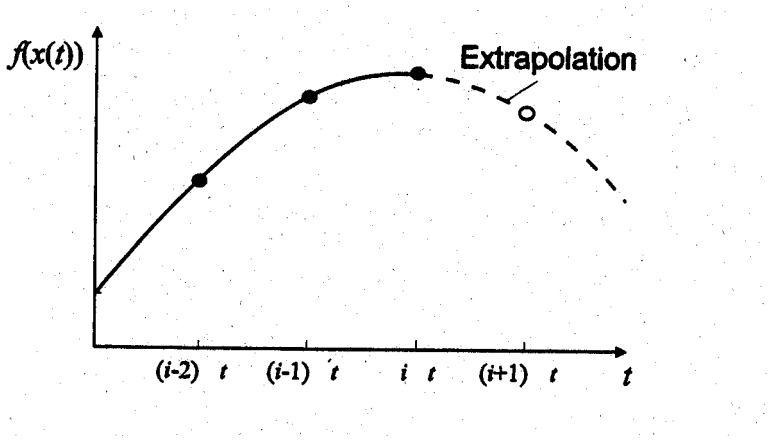


Abbildung 14 Extrapolation [3]

Eine Besonderheit der Mehrschrittverfahren ist die Startphase: Natürlich sind zu Beginn der Berechnung noch keine Werte, die zur Extrapolation genutzt werden können, bekannt. In der Regel werden die ersten Werte aus Einschrittverfahren, dann aus Zweischrittverfahren etc. gewonnen, bis das gewünschte Mehrschrittverfahren erreicht ist.

2.2.3.3 Methoden der räumlich verteilten zeitkontinuierlichen Simulation

Anders als für Probleme, die lediglich von der Zeit abhängig sind, sind räumlich verteilte Probleme zusätzlich von weiteren Dimensionen abhängig. Statt gewöhnlicher Differentialgleichungen führt dies zu Systemen *Partieller Differentialgleichungen (PDE)*.

2.2.3.3.1 Finite Differenzen

Die Behandlung dieser partiellen Differentialgleichungen läuft nach der Methode der finiten Differenzen analog zum Euler-Verfahren im gewöhnlichen Fall. Bei einem gegebenen Problem der Form: (*u* ist abhängig von *t,x,y,z* also: *u(t,x,y,z)*)

$$F(u, u_t, u_{tt}, \ldots, u_x, u_y, u_z, u_{xx}, u_{xy}, u_{xz}, \ldots) = 0$$

mit den entsprechenden Randbedingungen müssen die ersten, zweiten und weiteren Ableitungen durch Differenzenquotienten angenähert werden.

Für die erste Ableitung gibt es drei gängige Methoden:

- Vorwärtsreferenz:
$$u_x(t,x,y,z) \approx \frac{u(t,x+\Delta x,y,z)-u(t,x,y,z)}{\Delta x}$$
- Rückwärtsreferenz:
$$u_x(t,x,y,z) \approx \frac{u(t,x,y,z)-u(t,x-\Delta x,y,z)}{\Delta x}$$
- Symmetrisch:
$$u_x(t,x,y,z) \approx \frac{u(t,x+\Delta x,y,z)-u(t,x-\Delta x,y,z)}{2\Delta x}$$

Für Ableitung in die anderen Dimensionen gilt das obere analog.

Die zweite Ableitung wird angenähert durch:

$$u_{xx}(t,x,y,z) \approx \frac{u(t,x-\Delta x,y,z)-2u(t,x,y,z)+u(t,x+\Delta x,y,z)}{\Delta x^2}$$

Die gemischten Ableitungen werden genähert durch:

$$u_{xy}(t,x,y,z) \approx \frac{u_x(t,x,y-\Delta y,z)-u_x(t,x,y+\Delta y,z)}{\Delta y}$$

Die höheren Ableitungen werden analog gebildet.

Da die Randbedingungen nur auf dem rechtwinkligen Gitter definiert werden (genauer auf den Schnittpunkten des Gitters), ist es problematisch für das Verfahren der finiten Differenzen, wenn das vom System beschriebene Gebiet krummlinig umrandet ist.

2.2.3.3.2 Finite Volumina

Die Methode der finiten Volumina behandelt das Problem der krummlinigen Randflächen durch eine Triangulierung des gesamten Gebietes, also eine Aufteilung des Gebiets in Dreiecke (in 2D) oder Tetraeder (3D). Der krummlinige Rand wird durch den Polygonzug der Ersatzkörper genähert.

Die Methode wird angewendet für Probleme der Form

$$\nabla g(u) = f$$

Der Ansatz nutzt den Gauß'schen Integralsatz durch folgenden Ausdruck für alle Dreiecke V_i, mit *n* als Normale auf dem Rand:

$$\int_{V_i} \nabla g(u) \cdot dV = \int_{\partial V_i} g(u) \cdot n \cdot d\partial V = \int_{V_i} f \cdot dV$$

Die beiden rechten Integrale werden diskretisiert wie gehabt.

2.2.3.3.3 Finite Elemente

Die Methode der finiten Elemente ist das bekannteste Verfahren zur Lösung beliebiger partieller Differentialgleichungen. Gleichzeitig ist diese Methode allerdings methodisch und

rechnerisch sehr aufwändig. Die Grundideen des Verfahrens sollen in der Folge kurz erklärt werden.

Die Methode baut auf drei Grundideen:

1. Umwandlung in Variations- (Optimierungs-) Problem
2. Summenapproximation des Optimierungsproblems
3. „Pyramidenfunktionen“ auf der Triangulierung als Ansatzfunktionen.

Es lässt sich zeigen [56], dass sich beliebige Partielle Differentialgleichungssysteme in ein Optimierungsproblem umwandeln lassen. Optimierungsprobleme sind Probleme der Form:

Gesucht $u \big| E(u) = \min$

Dieses Optimierungsproblem wird durch eine Summenapproximation genähert:

$$u(x,y) \approx \sum_{i=1}^{n} a_i \cdot \phi_i(x,y)$$

Die ϕ_i sind Ansatzfunktionen, mit deren Hilfe das Optimierungsproblem auf ein hochdimensionales lineares Gleichungssystem zurückgeführt werden kann.

Die numerischen Eigenschaften dieses Gleichungssystems hängen entscheidend von der Wahl der Ansatzfunktionen ab. An dieser Stelle nutzt die Methode die Triangulierung des betrachteten Gebietes in finite Elemente (daher der Name des Verfahrens). Diese finiten Elemente (in 2D am einfachsten Dreiecke, Tetraeder in 3D) erlauben die Konstruktion so genannter „Pyramidenfunktionen“ (siehe Abbildung 15 Pyramidenfunktion in 2D und 3D).

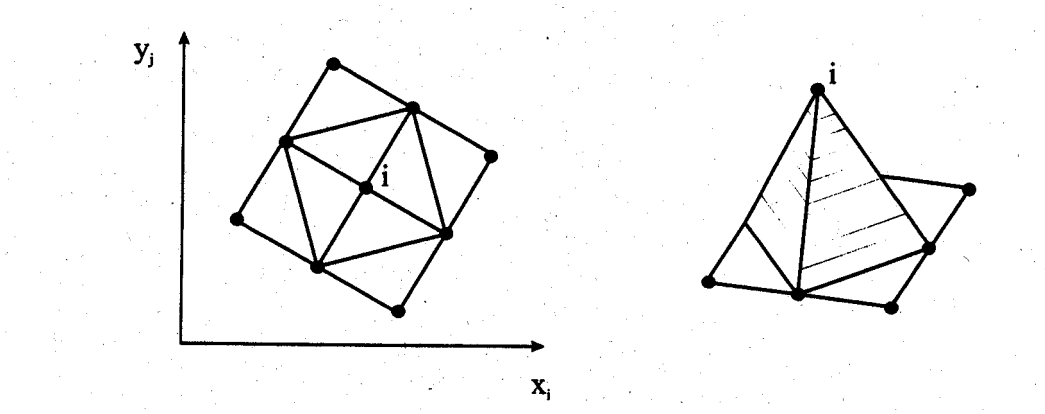


Abbildung 15 Pyramidenfunktion in 2D und 3D [3]

Diese Pyramidenfunktionen entkoppeln die Gleichungen für entfernt liegende Punkte des betrachteten Gebiets, was dazu führt, dass die Matrix des linearen Gleichungssystems schwach besetzt ist und die Form einer Tridiagonalmatrix ähnelt. Für diese Konfiguration existieren numerisch stabile und effiziente Lösungsalgorithmen [56].

2.2.4 Objektorientierte Simulation

Anders als die oben beschriebene Unterscheidung zwischen ereignisdiskreter und kontinuierlicher Simulation bildet die objektorientierte Simulation keine echte eigene Klasse

von Simulationen. Sie beschreibt eine konzeptionelle Herangehensweise bei der Erstellung eines Simulationsmodells. Das Vorgehen ist hier von aus dem Softwareengineering bekannten Techniken der objektorientierten Modellierung [8] geprägt.

Zentrale Elemente der objektorientierten Softwareerstellung, wie *Information Hiding*, *Polymorphismus* und *Vererbung*, werden hier für die Simulation aufgenommen. Der Grundgedanke objektorientierter Modellierung ist, ohne diese Bezeichnung zu erhalten und nicht mit allen der oben genannten Attribute ausgestattet, bereits seit langem in der Simulationstechnik bekannt.

Für die ereignisdiskrete Simulation ist der Übergang von einem schon immer durch *Entities,* also abgrenzbaren Einheiten, bestimmten Ansatz zu einem objektorientierten Ansatz recht einfach. Die kontinuierliche Simulation ist den Methoden der objektorientierten Modellierung weniger zugänglich. Ein Hauptmerkmal dieser Simulationstechnik ist das Bestreben, das gesamte System als geschlossenes Gleichungssystem darzustellen. Dies steht oft im Gegensatz zu objektorientierten Techniken. Am nächsten kommen hier Methoden der modularen Modellierung, also der Modellierung einzelner Teilaspekte eines Systems als Subsystem und der anschließende Aufbau des Gesamtsystems aus seinen Subsystemen. Vor der eigentlichen Simulation werden dabei jedoch die Gleichungen aus den Subsystemen zu einem gesamten, gemischt differential-algebraischen Gleichungssystem überführt und dann das gesamte System berechnet.

2.2.5 Multi-Agentenbasierte Simulation

Die Multi-Agentenbasierte Simulation hat in den letzten Jahren an Bedeutung gewonnen. Diese Entwicklung läuft parallel zu der zunehmenden wissenschaftlichen Bedeutung der Technologie intelligenter Software-Agenten im Bereich des Software-Engineering. [58].

Die Nutzung von Agententechnologien für die Simulation zeichnet sich allerdings durch zwei besondere Phänomene aus:

Zum einen rekrutieren sich die mit Hilfe dieser Simulationsmethode simulierten Systeme vorwiegend aus dem Bereich „künstliche Gesellschaften“, zum anderen werden die Methoden der Agententheorie in der Regel lediglich für die Modellierung der Systeme verwendet, jedoch nicht für die Simulationstechnik.

„Künstliche Gesellschaften“ beschreiben hier all jene Anwendungsbereiche, in denen sich die Modellierung des betrachteten Systems als Multi-Agenten-System quasi trivial aus der Struktur des Systems ergibt. Beispiele seien hier etwa [59], [60], die meisten Beispiele finden sich in der Soziologie [61].

Wie Drogoul et. al. [62] bemerken, zeigt sich bei der agentenbasierten Simulation das gleiche Phänomen, das auch für die objektorientierte Simulation beschrieben wurde: Zwar werden die Methodiken des Software-Engineering für die Modellierung des Systems benutzt, nicht jedoch für die tatsächliche simulationstechnische Umsetzung. Drogoul et. al. führen dies auf das besondere wissenschaftliche Interesse an der Modellierungstechnik zurück, das begleitet wird von einem gewissen Desinteresse an der nur für die Anwendungsdomäne interessanten Durchführung der eigentlichen Simulationsstudie.

2.2.6 Zusammenfassung

Aus den oben beschriebenen Zusammenhängen ergibt sich eine zusammenfassende Klassifikation von Simulations- und Modellierungstechniken.

Eine erste Einteilung kann nach der Art der Zeitachse erfolgen:

- kontinuierliche Zeitachse
- diskret äquidistante Zeitachse
- diskret nicht äquidistant Zeitachse.

Des Weiteren ist die Art der Zustandsbeschreibung wichtig:

- räumlich verteilte Systeme
- homogene Systeme

Der Zeithorizont der zu untersuchenden Dynamik lässt sich unterscheiden in

- stationäre und
- dynamische Systeme.

Über die oben beschriebenen Unterscheidungsmerkmale von Simulation und Modellierung hinaus lassen sich auch noch weitere Kriterien finden. Ebenso lassen sich die genannten Unterscheidungen weiter verfeinern. Eine weitere wichtige Anforderung an eine Simulationsstudie ist die Anforderung an die Visualisierung der Simulationsläufe und der Ergebnisse. Die wichtigste Unterscheidung hierbei ist, ob die Visualisierung

- offline oder
- interaktiv

stattfinden muss.

2.3. Visualisierung

2.3.1 Grundbegriffe

Als Visualisierung, genauer als wissenschaftlich-technische Visualisierung, wird die graphische Aufbereitung von Daten verstanden. Angelpunkt jeder Visualisierung ist das *Visualisierungsziel*. Das grundsätzliche Ziel jeder Visualisierung ist es, dem Betrachter das Auswerten zu ermöglichen oder zu vereinfachen. Die Wissenschaft [63] hat dieses grundsätzliche Ziel in drei Klassen von Visualisierungszielen differenziert:

- explorative Analyse
- konfirmative Analyse
- Kommunikation.

Im Fall der explorativen Analyse hat der Betrachter noch keine Hypothese gebildet, welche Strukturen oder Inhalte in den Daten zu finden sind. Ziel der Analyse ist es, diese Strukturen und Inhalte zu finden. Die konfirmative Analyse versucht, Hypothesen zu bestätigen oder zu widerlegen. Das Visualisierungsziel „Kommunikation“ nutzt die Mittel der Visualisierung, um Erkenntnisse an die Betrachter weiterzugeben. Die Quelle der Erkenntnisse, etwa aus explorativer oder konfirmativer Analyse mit Hilfe von Visualisierung, ist für die Visualisierung zur Kommunikation nicht relevant.

Nach Etablierung des Visualisierungsziels stellt sich die Frage, was eine „gute“ Visualisierung hinsichtlich des Ziels ist. Obwohl es keine allgemein gültige „beste“ Visualisierung gibt (hier sind nach wie vor Geschick, Erfahrung und Keativität des Autors der Visualisierung gefragt), existieren dennoch Kriterien, die eine gute Visualisierung erfüllen muss: Die Visualisierung

von Daten soll expressiv, effektiv und angemessen sein. Im Folgenden werden diese Begriffe näher erläutert.

Eine Darstellung ist *expressiv*, wenn die Daten unverfälscht wiedergegeben werden. Das ist die Grundlage für eine wissenschaftliche Auswertung. Eine verfälschte Darstellung könnte zu einer falschen Interpretation der Daten führen und somit zu falschen Annahmen oder Entscheidungen. Verfälscht kann hier zum Beispiel heißen, dass die gewählte Visualisierung Zusammenhänge zwischen Daten suggeriert, die in den Daten nicht vorhanden sind.

Die *Effektivität* der Datendarstellung gibt in hohem Maße den Nutzen der Visualisierung an. Eine Visualisierung ist *effektiv*, wenn alle visualisierten Merkmale – unter Berücksichtigung des Vorwissens des Betrachters – möglichst intuitiv wahrgenommen werden. Dabei sollten die Fähigkeiten des Betrachters und des Ausgabemediums optimal ausgenutzt sein. Der Aufwand für eine Visualisierung kann mit den Ansprüchen an Effektivität schnell wachsen. Daher ist es wichtig, das Visualisierungsvorhaben auf Angemessenheit zu prüfen. *Angemessen* heißt, dass der Aufwand beim Erstellen in einem guten Verhältnis zum Nutzen steht. Was ein gutes Verhältnis ist, muss der Autor der Visualisierung entscheiden, da vor allem die Kosten für das Erstellen der Visualisierung eine wichtige Rolle spielen. Angemessenheit ist für die Praxis also ein wichtiges Kriterium [63].

Der Vorgang beim Erstellen einer Visualisierung lässt sich in drei wesentliche Schritte unterteilen:

- Filtering
- Mapping
- Rendering [63].

Diese Schritte werden im Allgemeinen sequentiell abgearbeitet (siehe Abbildung 16 Filtering – Mapping – Rendering). Die Rohdaten, also jene Daten, die durch Messungen, Erhebungen oder aus anderen Quellen entstehen, werden beim *Filtering* aufbereitet. Ein Ziel des Filtering kann dabei sein, unerwünschte oder fehlerhafte Daten zu eliminieren. Aber auch das Ergänzen von Daten, zum Beispiel durch eine Voronoi-Zerlegung [66], gehört zum Filtering. Wichtig ist dabei, dass das Filtering kein automatischer Prozess sein muss; um Fehler oder Muster zu erkennen, kann Interaktion erforderlich sein. Weitere mögliche Filter sind: Werte glätten, Daten oder Einheiten konvertieren, Verzerrungen beseitigen, weitere Werte durch Interpolation berechnen.

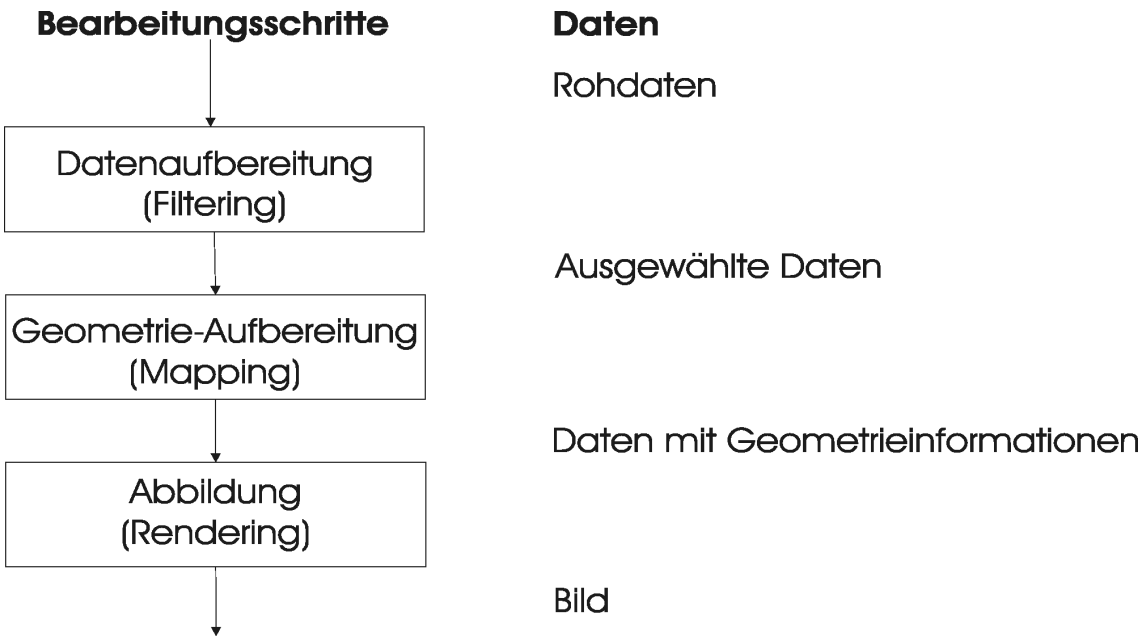


Abbildung 16 Filtering – Mapping – Rendering (nach [63])

Beim zweiten Schritt, dem *Mapping*, werden die aufbereiteten Daten auf ein geometrisches Modell abgebildet. Wenn die Daten selbst Geometrieinformationen enthalten, können diese genutzt werden, müssen es aber nicht. Das Mapping ist der wichtigste Schritt der Visualisierung; hier wird festgelegt, wie das Bild schließlich aussehen wird. Am Ende dieses Schrittes erhält man geometrische Daten mit Darstellungsinformationen wie Farben, Muster etc. Daraus wird beim *Rendern* das endgültige Bild (bzw. bei Animationen Bilder) erzeugt. Das Rendering ist also das Sichtbarmachen der beim Mapping erzeugten geometrischen Informationen. Dabei bestimmen unterschiedliche Ziele das eingesetzte Verfahren. Ziel kann es zum Beispiel sein, die Daten möglichst realistisch wiederzugeben oder nach bestimmten Gesichtspunkten zu abstrahieren. Wenn eine Animation erzeugt werden soll, so müssen mehrere Bilder gerendert und hintereinander wiedergegeben werden.

Bei einer Visualisierung kann zwischen dem Autor und dem Betrachter unterschieden werden. Der Autor ist der Experte, der die Rohdaten zur Verfügung stellt. Außerdem erzeugt er die Visualisierung. Der Betrachter ist der Leser der visuell aufbereiteten Daten. Er schaut zu und versucht, aus der Visualisierung zu lernen. Die Trennung der Rollen des Autors und des Betrachters sind in der Visualisierungspipeline fließend.

Beispielsweise kann ein Autor den gesamten Prozess der Visualisierung übernehmen und dem Betrachter ein fertiges Bild präsentieren. Der Betrachter hat keinen Einfluss auf das Bild.

Eine andere Art der Visualisierung entsteht, wenn der Autor die Visualisierungspipeline nur bis zum Mapping bearbeitet. Der Betrachter kann dann das erzeugte Bild beeinflussen und damit in der Visualisierung navigieren.

Eine dritte Variante entsteht, wenn der Autor nur die Daten liefert und der Betrachter mit einem zur Verfügung gestelltem Visualisierungssystem das Mapping sowie die Darstellung selbst übernehmen kann.

Es ergeben sich damit verschiedene Interaktionsmöglichkeiten, die eine Visualisierung bzw. ein Visualisierungssystem zur Verfügung stellt.

2.3.2 Rolle der Visualisierung im Kontext von Simulation und Modellierung

Bereits im Kaptitel zur Simulation wurde die Notwendigkeit der Ergebnispräsentation, also der Visualisierung von Ergebnissen, erwähnt. Dabei wurde stillschweigend davon ausgegangen, dass für die notwendigen Zwischenschritte, insbesondere für die Simulationsmodelle, eine angemessene Darstellung vorhanden ist. Diese Annahme lässt sich jedoch nicht in allen Fällen einfach aufrechterhalten, hat man es hier doch mit nicht-linearen, komplexen dynamischen Systemen zu tun. Die Erforschung von komplexen Systemen ist heute Bestandteil nahezu jeder wissenschaftlichen Disziplin, nicht nur im naturwissenschaftlichen Bereich, etwa in der Physik, Chemie, Medizin oder, wie in dieser Arbeit, in der Biologie, sondern auch in den Geisteswissenschaften, etwa in der Psychologie, Soziologie oder Ökonomie.

Entsprechend ist zu beachten, dass die „Computersimulation nicht nur eine neue Methode in den Wissenschaften [ist], sondern auch ein neues Symbolsystem, das auf einem neuen Typus von Schriftlichkeit basiert – der digitalen Schrift“ [74].

Simulationsmodelle treten damit neben den an Sprache orientierten Schriftbegriff und die formale semiotische Schrift, also ein System von abstrakten Zeichen, z. B mathematische Formelzeichen, Strukturformeln der Chemie etc. Mit dieser Rolle werden auch Kommunikation, Austauschbarkeit und Verifizierbarkeit von Simulationsmodellen besonders wichtig. Das Simulationsmodell wandelt sich hierbei vom Mittel zum Zweck der Erkenntnisgewinnung hin zum eigentlichen Ergebnis der Forschung.

„Das größte Handicap dieser [neuen digitalen] Schrift ist ihre Unanschaulichkeit, denn sie ist [a priori] nicht mehr visuell repräsentiert“ [74]. Eine Visualisierung ist demnach notwendig.

2.3.3 Methoden der Visualisierung

Wie oben erwähnt, ist vor allem das Mapping, abgesehen vom Filtering und der damit verbundenen Selektion der Daten, der entscheidende Schritt beim Erstellen einer Visualisierung. Die visuellen Variablen, auf die die Daten abgebildet werden, sind dabei ausschlaggebend. Beispiele für visuelle Variablen sind Geometrie und Farbe.

Beim Mapping auf Geometrie können Daten vor allem auf die Variablen Position, Größe und Form abgebildet werden. Beispielsweise sind Diagramme wie Balken-, Kreis- oder Flächendiagramme typische Anwendungen des Mapping auf Geometrie. Bei einem Balkendiagramm werden verschiedene Daten (ordinale Daten) auf verschieden große und entsprechend positionierte Rechtecke abgebildet. Die Auswahl der Geometrie, auf die die Daten abgebildet werden sollen, ist für die Expressivität und die Effektivität der Visualisierung entscheidend. Oft liegt es in der Natur der Daten, dass eine bestimmte Art der geometrischen Abbildung nahe liegt.

Beim Mapping auf Farbe werden Farben verwendet, um Informationen darzustellen. Farbe kann beispielsweise zur Unterscheidung von qualitativ verschiedenen Daten oder zur Betonung von bestimmten Daten verwendet werden. Weiterhin können mit Farben Assoziationen zwischen Daten hergestellt werden. Sollen Farben zur Unterscheidung (nominaler Daten) verwendet werden, so müssen optisch gut unterscheidbare Farben benutzt werden. Weiterhin sollte die Zahl der verwendeten Farben nicht zu groß sein, da sonst der Betrachter keine Zuordnung der Daten mehr vornehmen kann. Farben können auch benutzt werden, um eine Übersicht über ordinale Daten zu geben. Dazu werden Farbskalen verwendet, die Quantität auf Farbe abbilden

(siehe Abbildung 17). Die Temperatur-Farbskala scheint dafür besonders gut geeignet zu sein, da sie als natürlich empfunden wird und somit eine intuitiv richtige Interpretation beim Betrachter bewirkt.

Abbildung 17 Farbskalen a) Temperatur-Farbskala b) Magenta Farbskala [63]

Unterscheidungen können statt mit Farbe auch mit Schraffuren (allgemein durch Texturen) vorgenommen werden.

Häufig werden mehrere visuelle Variablen benutzt, um Daten darzustellen. Selbst einfache Diagramme bedienen sich neben Geometrien auch des Mappings auf Farbe, um Unterscheidungen vorzunehmen.

Ein visuelles Objekt kann aus mehreren verschiedenfarbigen geometrischen Primitiven aufgebaut sein und kann damit komplexe Daten repräsentieren.

2.3.4 Visualisierungstechniken

Abgesehen vom Abbilden der Werte oder der Art der Daten auf die visuellen Variablen, wie Position, Größe, Form und Farbe, können noch weitere Eigenschaften der Daten visualisiert werden.

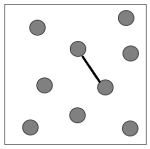


Abbildung 18 Konnektivität

Konnektivität

Bei der Visualisierung von Konnektivität werden Beziehungen oder Zusammenhänge der Daten dargestellt. Dabei reicht z. B. eine einfache Linie aus, um zwei Punkte visuell miteinander zu verbinden (siehe Abbildung 18 Konnektivität). Die Grundlage zur Darstellung der Konnektivität ist der Graph. Ein Graph besteht aus Knoten, die ein Datum mit den oben beschriebenen Methoden repräsentieren. Die Knoten sind über Kanten miteinander verbunden. Kanten können wiederum Attribute und/oder eine Richtung haben, die über Farbe, Textur, Beschriftung oder Form visuell dargestellt werden können.

Raumbezogene Daten

Wenn Daten aus Messungen oder durch Simulationen berechnete Werte einen Raumbezug haben, so werden diese Daten an konkreten Orten gewonnen. Das räumliche Bezugssystem kann punktuell, lokal oder global sein, entsprechend der Natur der Daten. Der Wirkungskreis der Daten muss bei der Darstellung der raumbezogenen Daten mit berücksichtigt werden. Zur Darstellung raumbezogener Daten wird der Messraum (meist zwei- oder dreidimensional) auf das durch die Visualisierung zu erzeugende Bild übertragen. Dabei werden bei der direkten Darstellung die Daten an den entsprechenden Punkten visualisiert. Bei der indirekten Darstellung werden nur die Messpunkte angezeigt. Die Selektion eines Messpunktes hat eine separate Auflistung der Messwerte zur Folge. Komplexere multivariate Daten können auch als Kombination von direkter und indirekter Darstellung des Raumbezuges visualisiert werden. In diesem Fall sind z. B. die Informationen mit der stärksten Aussagekraft direkt visualisiert, während zusätzliche Daten nur auf Abruf verfügbar werden.

In jedem Fall ist es notwendig, dem Betrachter eine gute Navigation durch den visualisierten Messraum zu ermöglichen.

Zeitbezogene Daten

Häufig haben Daten, die visualisiert werden sollen, einen Zeitbezug. Da Zeit keine beliebige, unabhängige Variable ist, kommt ihr eine spezielle Rolle bei der Darstellung zu. Als Erstes muss zwischen dem Zeitbezug der Daten und dem Zeitbezug der Darstellung unterschieden werden. Die Daten können statisch zu einem bestimmten Zeitpunkt gewonnen worden sein. Sind Daten vorhanden, die von verschiedenen Zeitpunkten stammen, so ist der Zeitbezug der Daten dynamisch. Es wird unterschieden, ob die Daten kontinuierlich, also für jeden Zeitpunkt des betrachteten Zeitraumes, vorliegen oder zu diskreten Zeitpunkten gehören, wobei kontinuierliche Daten auch aus Berechnungen oder Simulationen hervorgehen können. Bei Einhaltung der durch das Abtasttheorem [187] gegebenen Randbedingungen lassen sich aus den abgetasteten Daten die kontinuierlichen Signale rekonstruieren.

Die Darstellung der Daten kann durch eine statische oder eine dynamische Repräsentation erfolgen. Die dynamische Repräsentation kann linear oder nichtlinear sein. Bei Linearität läuft die Zeit stetig und mit gleich bleibender Geschwindigkeit ab. Die Geschwindigkeit kann allerdings variieren von Zeitlupe über Echtzeit zu Zeitraffer. Eine nichtlineare Darstellung springt in der Zeit, was im Allgemeinen ein Springen der Daten zur Folge hat. Das kann unbeabsichtigt die Aufmerksamkeit des Betrachters derart auf sich ziehen, dass die wesentlichen Informationen nicht mehr wahrgenommen werden (Verletzung der Effektivität und Effizienz!)

Dadurch ergeben sich mehrere Möglichkeiten der Kombination von Darstellungen, abhängig vom Zeitbezug der Daten, wie sie in Tabelle 2 angegeben sind.

Zeitbezug der Daten / Zeitbezug der Darstellung	Statisch	Diskret abgetastet	Dynamisch
Statische Repräsentation	Zeitangabe	spezielle Ikonen; Mehrfenster-Technik; Zeitangabe	Zeitdiagramme
Dynamische Repräsentation	Kamerafahrten; Objektbewegung; Deformationen	sequenzielle Abfolge von Darstellungen;	sequenzielle Abfolge von Darstellungen;

Tabelle 2 Zeitbezüge

Beispielsweise können zu diskreten Zeitpunkten abgetastete Messwerte statisch repräsentiert werden, indem spezielle Ikonen, wie z. B. stilisierte Uhren, dargestellt werden. Eine dynamische Repräsentation von dynamischen Daten kann durch eine sequenzielle Abfolge von Visualisierungen der einzelnen Zeitpunkte dargestellt werden.

Sollen die Werte der Daten abgelesen werden, eignen sich eher eine statische Repräsentation oder ein Standbild der dynamischen Repräsentation.

2.3.5 Dreidimensionale Visualisierung

2.3.5.1 3-D-Repräsentationsmethoden

Um ein dreidimensionales Objekte in einem 3-D-Computermodell darstellen zu können, muss das Aussehen dieses Objekt zuerst durch ein mathematisches Modell beschrieben werden, um anschließend durch eine geeignete Datenstruktur im Computer repräsentiert werden zu können. Es gibt unterschiedliche Repräsentationsmethoden, die für verschiedene Problemstellungen verwendet werden können. Kriterien, nach denen diese Methoden unterschieden werden können, sind Genauigkeit, Anwendungsbreite, Verifizierbarkeit, Ressourcenbedarf, Abgeschlossenheit und Erweiterbarkeit [64]. Zwei dieser Methoden, die Repräsentation durch Elementarobjekte und die Repräsentation über Flächenmodelle, wurden bei der 3-D-Darstellung der Zelle verwendet und werden nun kurz erläutert.

2.3.5.1.1 Elementarobjekte

Objekte können in ihrem Aussehen häufig hinreichend genau durch das Zusammensetzen unterschiedlicher als Elementarobjekte [65] oder vordefinierte Primitive [64] bezeichnete Körper oder Formen modelliert werden. Diese Elementarobjekte sind in der Regel einfache Körper, wie Ellipsoid, Quader, Zylinder oder Kegel, können aber auch beliebig komplex sein. Sie sollten allerdings durch wenige Parameter beschrieben werden können. In Abbildung 19 ist ein aus Elementarobjekten modelliertes Modell eines Schreibtischstuhls dargestellt.

2.3.5.1.2 Flächenmodell

Bei der Flächenmodelldarstellung wird das Objekt durch eine Kombination aus analytischen oder approximierten Flächen (Bezier- oder Splineflächen) beschrieben [65]. In der computergrafischen Darstellung werden diese Flächen oft durch konvexe Polygone nachgebildet. Jedes dieser Polygone wird durch Eckpunkte definiert, die durch Kanten miteinander verbunden werden. Die Darstellung der Kantenzüge ohne das Zeichnen der Flächen wird als Drahtgitterdarstellung bezeichnet (siehe Abbildung 20).

Abbildung 19 Ein aus Quadern, Kugeln und einem Zylinder zusammengesetztes Modell eines Schreibtischstuhls

Abbildung 20 Drahtgitter- und Flächenmodell-darstellung eines Keramikgefäßes [182]

Damit das Flächenmodell vollständig beschrieben ist, muss noch angegeben werden, welche Seite einer Fläche die Innen- bzw. die Außenseite des modellierten Objektes definiert. Dafür wird für jede Fläche ein Normalvektor bestimmt, dessen Richtung die Außenseite der Fläche ausweist.

2.3.5.1.3 Koordinatentransformationen

Für die Darstellung einer dreidimensionalen Szene müssen die 3-D-Koordinaten der Objekte der Szene auf einer zweidimensionalen Ausgabefläche abgebildet werden. Diese Abbildung kann durch das Anwenden verschiedener hintereinandergeschalteter Matrixmultiplikationen auf die Objektkoordinaten durchgeführt werden.

Jede der hintereinandergeschalteten Matrizen ist für eine bestimmte Koordinatentransformation zuständig. In Abbildung 21 ist die Koordinatentransformations-Pipeline für eine Rendering-Operation einer Szene skizziert. Jede Transformation bildet die Koordinaten eines bestimmten Koordinatensystems auf ein anderes ab. Die in der Abbildung dargestellten unterschiedlichen Koordinatensysteme werden nun kurz erläutert (siehe dazu [57]):

- Objektkoordinaten: Um Objekte im Raum relativ zu anderen Objekten rotieren, bewegen oder skalieren zu können, werden für die zu transformierenden Objekte eigene Koordinatensysteme definiert. Durch die Modelmatrix werden die zu dem Objektkoordinatensystem relativen Koordinaten in ein absolutes Weltkoordinatensystem transformiert.
- Weltkoordinaten: Die Objekte der Szene werden alle in einem absoluten Weltkoordinatensystem angeordnet. In diesem Koordinatensystem sind alle Objekte bereits positioniert und orientiert, das heißt, in diesem Koordinatensystem ist das fertige Bild der Szene gespeichert.
- Kamerakoordinaten: Um ein Bild der Szene auf eine Fläche projizieren zu können, muss zuerst bestimmt werden, welcher Teil der Szene projiziert werden soll. Dazu wird die Position einer Kamera (bzw. eines Betrachters) in Bezug auf die Szene festgelegt, wobei das Weltkoordinatensystem in ein Kamerakoordinatensystem transformiert wird.
- Projektionskoordinaten: Nachdem die Position der Kamera definiert wurde, kann die Szene nun auf eine zweidimensionale Fläche projiziert werden. Dafür gibt es unterschiedliche Methoden, zwei davon werden nun kurz erläutert (siehe Abbildung 22) [64]:
  - Orthogonalprojektion: Bei der Orthogonalprojektion stehen die Projektionsstrahlen parallel zur Bildebene, das Projektionszentrum liegt im Unendlichen.Durch diese Art der Projektion wird ein unrealistisches Bild der Szene gezeichnet, da die Informationen über die Entfernung eines Objektes verloren gehen. Da bei der orthogonalen Projektion die Längenverhältnisse der Seiten bestehen bleiben, wird diese Projektionsmethode häufig für technische Zeichnungen verwendet.
  - Perspektivische Projektion: Bei der perspektivischen Projektion befindet sich das Projektionszentrum im Endlichen. Durch diese Projektionsmethode werden Abstände, die weit vom Projektionszentrum entfernt sind, kleiner dargestellt als Abstände, die entsprechend nahe am Zentrum liegen, wodurch bei der Darstellung Tiefeninformationen mitvermittelt werden. Die perspektivische Projektion wird für die realistische Darstellung einer Szene eingesetzt.
- Window-Koordinaten: Durch die Projektion werden die Kamerakoordinaten auf eine Projektionsfläche transformiert. Diese Fläche befindet sich selbst im dreidimensionalen Raum und besitzt normalerweise nicht die Ausmaße der Ausgabefläche (Viewport), auf dem die Szene dargestellt werden soll. Die letzte Transformation über die Viewport-Matrix passt nun das Bild der Projektionsfläche auf die Größe des Viewports an. Die dadurch errechneten Koordinaten werden als Window-Koordinaten bezeichnet.

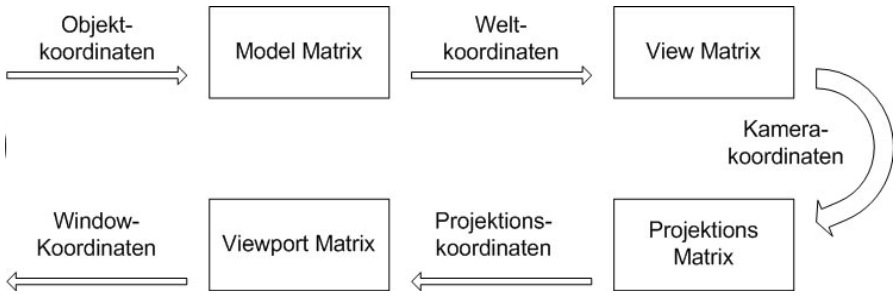


Abbildung 21 Koordinatentransformations-Pipeline

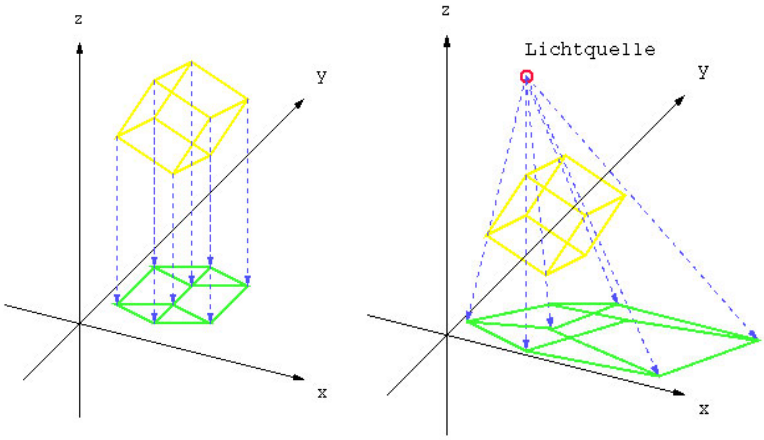


Abbildung 22 Darstellung einer orthogonalen (links)
und einer perspektivischen (rechts) Projektion [64]

2.3.5.2 Szenengraphen

Eine hierarchisch angeordnete dreidimensionale Szene lässt sich durch einen gerichteten azyklischen Baum, einen so genannten Szenengraphen, darstellen [75].

Der Szenengraph besteht aus Knoten (Node), die über Kanten miteinander verbunden sein können, und Blättern (Leaf), die sich an den Knoten befinden. Die Informationen über die Geometrien und Materialeigenschaften der zeichenbaren Objekte stecken in den Blättern der Knoten. Die Knoten werden dazu verwendet, um Objekte zu gruppieren oder zu positionieren; durch die Kanten werden die logischen Zusammenhänge zwischen den Grafikobjekten festgelegt. Der Pfad von derWurzel bis zu den Blättern enthält alle Informationen über das zu rendernde Objekt (siehe Abbildung 23).

Unterschiedliche Szenengraph-Modelle können verschiedene Knotentypen besitzen. Übliche Knotentypen sind [76]:

- Geometrieknoten: Dieser Knotentyp enthält Leaf-Objekte, die die Form eines Objektes beschreiben.
- Kompositionsknoten: Um mehrere Objekte logisch zu gruppieren oder zu ordnen, werden Kompositionsknoten eingesetzt.

- Platzierungs- und Dimensionierungsknoten: Über diesen Knotentyp können für bestimmte Objekte lokale Objektkoordinatensysteme definiert werden. Dadurch lassen sich die Objekte relativ zu anderen Objekten der Szene bewegen und drehen. Normalerweise sind Platzierungs- und Dimensionierungsknoten ebenfalls Kompositionsknoten, das heißt, sie können ganze Teiläste des Szenengraphen manipulieren.

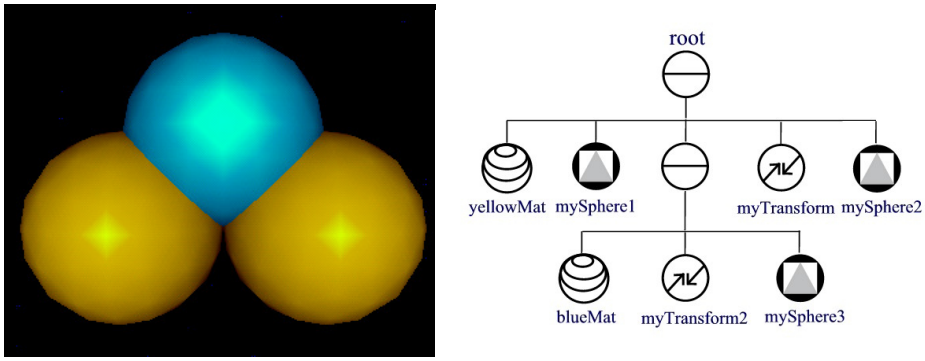


Abbildung 23 Szenengraph (rechts) und die dazugehörige Darstellung einer Szene [178]

- Erscheinungsbild-Knoten: Diese Knoten enthalten Leaf-Objekte, die das äußere Erscheinungsbild des Objektes bestimmen, wie z. B. Materialeigenschaften, Beleuchtungs- und Spiegelungseffekte oder Texturen.
- Knoten für globale Einstellungen: Knoten für globale Einstellungen beschreiben Rendering-Informationen, die für die komplette Szene von Bedeutung sind, wie z. B. Umgebungslicht- oder Hintergrund-Einstellungen.
- Knoten für nicht-visuelle Informationen: Solche Knoten werden für besondere Effekte eingesetzt, die nicht durch den Rendering-Mechanismus erfasst werden, wie z. B. Soundeffekte und Musik.
- Aktionsknoten: Der Szenengraph an sich ist ein statisches Gebilde. Damit sich der Aufbau des Graphen dynamisch ändern kann, werden Aktionsknoten definiert, die Benutzerinteraktionen oder Animationen ermöglichen.

In [164] sind vier Gründe aufgelistet, die für einen Einsatz von Szenengraphen bei der Entwicklung von grafischen Anwendungen sprechen:

- Performanz: Durch die Anordnung der grafischen Informationen in einem Szenengraphen können durch bestimmte Techniken, wie z. B. das Gruppieren und gleichzeitige Zeichnen von Objekten mit gleichen Materialeigenschaften, doppelte Zeichenoperationen vermieden und demzufolge die CPU entlastet werden.
- Produktivität: Durch die Verwendung von Szenengraphen kann der Entwickler auf einer höheren Abstraktionsebene programmieren. Anweisungen, die bei Hardware-naher Programmierung mehrere tausend Code-Zeilen umfassen würden, können in der Szenengraph-Ebene eventuell durch wenige Code-Zeilen realisiert werden.
- Portabilität: Die Beschreibung von Szenengraphen kann portierbar gestaltet werden, um die Verwendung der Szenengraphen auf verschiedenen Plattformen zu ermöglichen.

- Skalierbarkeit: Durch die Verwendung von Szenengraphen wird die Aufteilung der Rendering-Operationen auf mehrere Hardwareplattformen und die Verwaltung der Hardware-Cluster vereinfacht.

Szenengraphen werden in unterschiedlichen Anwendungsbereichen eingesetzt. Die Metasprache VRML basiert auf dem Szenengraph-Konzept wie auch die Java3D API [190]. Unter C++ gibt es mehrere OpenGL-basierte Szenengraph-APIs, wie OpenGL Optimizer [161], Open GL Performer [191], OpenSG [192] oder OpenSceneGraph [164].

2.3.6 Zusammenfassung

In diesem Kapitel wurde die Visualisierung als Mittel des wissenschaftlichen Erkenntnisgewinns eingeführt. Nach einer Einführung zur Visualisierung, in der wichtige Grundbegriffe dargelegt wurden, erfolgte die Einordnung der Visualisierung in den Kontext von Simulation und Visualisierung. Es wurde beschrieben, dass Visualisierung eine wichtige Ergänzung zur Simulation darstellt. Grundlage hierfür ist die Tatsache, dass die „Sprache der Simulation“, genauer die Sprache, mit der Simulationsmodelle verfasst werden, wenig anschaulich ist. Visualisierung muss hier eine helfende Rolle spielen.

Die Methoden und Anwendungsgebiete der Visualisierung sind zu vielfältig, als dass sie im Rahmen dieser Arbeit vollständig darzustellen wären. Aus diesem Grund wurden grundlegende Methoden und Techniken der Visualisierung beschrieben. Für die weiteren Konzepte und Realisierungen, die in dieser Arbeit beschrieben werden, sind jedoch insbesondere grundlegende Techniken der 3-D-Visualisierung von Relevanz. Entsprechend wurden die Grundtechniken der Polygonaldarstellung aufgezeigt. Darauf aufbauend wurde die Technik der Szenengraphen beschrieben.

2.4. Interaktion

In diesem Kapitel sollen grundlegende Erkenntnisse zur *Interaktion* zusammengefasst werden. *Interaktion* ist hier die Kurzform von Mensch-Maschine-Interaktion (*MMI*, *Human Computer Interaction, HCI*) und beschreibt all jene Aspekte von Hard- und Software, die sich auf den Berührungspunkt, die *Schnittstelle*, zwischen dem menschlichen *Benutzer* einer Maschine und eben dieser *Maschine* bezieht. Obwohl sich der Terminus *Maschine* auf Maschinen jeder Art beziehen kann, ist in diesem Kontext selbstverständlich der Computer gemeint.

Die Geschichte der MMI ist so alt wie die Geschichte des Computers, so alt wie die Probleme des Benutzers, seine „Wünsche“ oder Befehle dem Rechner „mitzuteilen“ und dessen Ergebnisse zu verstehen. Die (historisch) ersten Benutzer eines Computersystems waren eben jene Menschen, die auch die erste Software erstellten, die Programmierer. Folgerichtig waren die ersten Anstrengungen im Bereich MMI auf diese Zielgruppe ausgerichtet. Allerdings gab es bereits in den frühen 1960er Jahren Überlegungen, die über diese Sichtweise hinausgingen. Für die hier vorliegende Arbeit sind die mäandrischen Entwicklungen der MMI in den 1960er und frühen 1970er Jahren nur insofern interessant, als sie die Entwicklung der Maus [67], [68] und anderer Geräte, wie Lightpens etc., als Eingabegerät neben der Tastatur und die Entwicklung vom Batchbetrieb, z. B. mittels Punchcards, hin zur direkten Terminaleingabe brachte.

Eine signifikante Weiterentwicklung war die von Xerox PARC 1983 kommerziell eingeführte Technik des *Graphical-User-Interfaces (GUI)* [69].

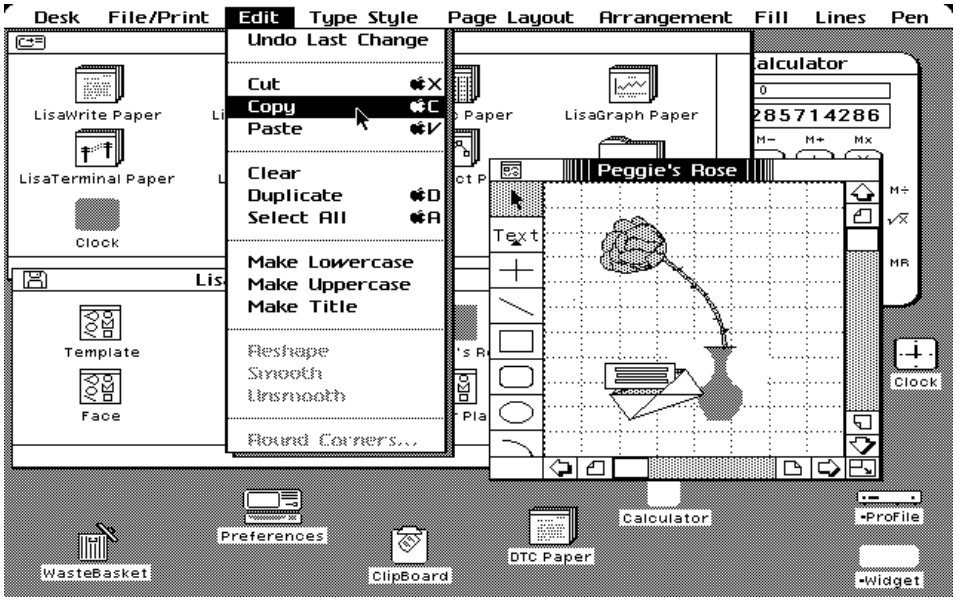


Abbildung 24 GUI Apple [193]

Mit Hilfe der Maus und der Tastatur als Eingabegeräten und des Monitors als Ausgabegerät war es nun möglich, in verschiedenen Fenstern verschiedene Anwendungen laufen zu lassen. Dieser Ansatz stellte einen enormen Zuwachs der Kommunikationsfähigkeit des Computers mit dem Benutzer dar, weil die Bandbreite des visuellen Kanals sehr viel stärker ausgenutzt werden konnte. Neu war ebenfalls der Einsatz von Ikonen, welche die Art einer Anwendung oder einer Aufgabe sehr viel besser beschreiben können als eine reine Textinformation. Zusätzlich müssen Befehle nicht mehr auswendig gelernt werden, da ein weiteres Konzept eingebaut ist: Menüs. Zwar war diese Technik nicht mehr neu, unterstützte die Bedienung einer Anwendung jedoch ungemein. Fenster und Menüs wurden zum Bestandteil des Betriebssystems, und der Benutzer musste sich nicht mehr an verschiedene Menüstrukturen von verschiedenen Anwendungen anpassen. Es gab einen übergreifenden Zugang zu jeder Anwendung. Diese Charakteristika führten dazu, dass für derartige GUIs auch der Begriff WIMPs üblich wurde (Windows, Icons, Menues, Pointing (Devives)).

Als Eingabemetapher wurde oft ein Pfeil-Symbol gewählt, welches das Zeigen auf Objekte im Anzeigebereich symbolisieren soll. Für das Klicken von 2-D-Tasten und einfache Auswahlaufgaben ist dies eine gute Methode, da für die eigentliche Aktion, das Positionieren und Klicken, ein sehr gutes Mapping möglich ist. Zudem erleichtert der Einsatz von Mauszeigern die Einarbeitungszeit für Anfänger enorm, da die Maus relativ einfach zu bedienen ist.

Trotz aller Vorteile gegenüber der Kommandozeile hat ein WIMP-Interface auch Nachteile. So bedarf diese Art von Schnittstelle einiger Gewohnheit und eines Mindestmaßes an Verständnis für die zugrunde liegenden Aufgaben. Ein generelles Problem von Computern ist die Undurchschaubarkeit der internen Prozesse für den unerfahrenen Benutzer. Der Klick auf einen Schalter und die folgende Reaktion des Computers sind nicht immer leicht in Zusammenhang zu bringen und schrecken viele Leute ab. Die Führung des Benutzers durch Menüs ist eine fehlerträchtige Angelegenheit, weil der Benutzer den Überlegungen des Entwicklers ausgeliefert ist. Da Fenster und somit die zugehörigen Menüs keine feste Position und nicht einmal eine feste Größe haben, kann sich der Benutzer nur mühsam ein konzeptionelles Modell des Systems bilden. Des Weiteren stehen gleichnamige Menüeinträge in verschiedenen

Anwendungen für verschiedene Aktionen, alle von abstrakter Natur. Dem Anspruch der Determiniertheit von Benutzungselementen wird dieses Paradigma der *Wi*ndows, Icons, *M*enues and *P*ointer (*WIMP*) nicht gerecht.

Der Einsatz von Metaphern verstärkt u. U. die vorhandene Verwirrung zusätzlich, da durch die Namensgebung und das Aussehen der Metapher eine Erwartungshaltung geweckt wird, welche jedoch die Maus und der Zeiger als Eingabegeräte nicht bedienen können. Ein Beispiel: Während der Informatiker sich über die Verwahrung alter Dateien im Papierkorb freut, ist für den unerfahrenen Benutzer schon das abstrakte Konzept der Datei nicht zu verstehen. Ganz zu schweigen von der Speicherung von Dateien im Papierkorb, der mit einem Klick geöffnet und mit einem Klick geleert wird.

Damit sind wir bei der diskreten Methode der Bedienung: Ein Klick reicht zum Auslösen einer Aktion, ganz gleich ob Formatieren der Festplatte oder Wechseln zum nächsten Fenster, wodurch Fehlbedienungen immer wieder vorkommen. In erster Linie liegt dies an der schwierig positionierbaren Maus und den eng beieinander platzierten Buttons. In zweiter Linie werden die Verhaltensweisen des Menschen außer Acht gelassen. Der Mensch generalisiert und liest nicht jede Warnung in jedem Fenster.

Bei dieser diskreten Art der Steuerung sind noch weitere Fehlbedienungen zu bemängeln. Das Benutzen eines Werkzeugs in der realen Welt ist ein kontinuierlicher Prozess, über den man in den meisten Fällen die Kontrolle hat, sei es das Mischen von Farben oder das Drehen einer Schraube. Bei WIMP-Systemen ist Einflussname auf ausgelöste Aktionen oft nicht möglich; falls doch, ist es immer von der Arbeit der Entwickler abhängig, ein Logbuch der durchgeführten Aktionen zu führen, um diese rückgängig machen zu können. Von der Tatsache abgesehen, dass diese Methoden oft ihre Fehler haben, ist es doch eine Bekämpfung der Symptome. Auch die manchmal unausweichlichen „Klickorgien“ sind sehr konzentrationsstörend. Der Benutzer wird von der eigentlichen Aufgabe abgelenkt, weil die Ausrichtung des Zeigers explizit erfolgt. Die Bewegungen des Zeigers sind so ungenau, dass keine kognitive Karte der Oberfläche aufgebaut werden kann, welche eine intuitive Benutzung zulassen würde.

Auch das Feedback des Systems ist unzureichend. Zwar kann durch die bessere Ausnutzung der Bandbreite des visuellen Kanals deutlich mehr Information vermittelt werden, das Feedback beschränkt sich im besten Fall aber auf einen Fortschrittsbalken, dessen Verweilen in einer bestimmten Position jedoch nicht bedeutet, dass keine Berechnungen mehr im Hintergrund stattfinden. Die meisten Aktionen liefern nur Feedback, indem ein Ergebnis präsentiert wird. Man erfährt nicht, ob die richtige Taste geklickt wurde, ob die Aktion rückgängig gemacht werden kann oder ob die Aktion noch gar nicht beendet ist und man nur ein Zwischenergebnis präsentiert bekommt.

Vergessen werden darf auch nicht, dass es sich hier um ein 2-D-System handelt, für dessen Bedienung die Freiheitsgrade der Hand reduziert werden müssen auf die beschränkten Möglichkeiten des Window-Systems – erst durch die Maus, dann durch den Zeiger. Auch der Einsatz von Ikonen und Farben ist von großer Bedeutung, zumindest für erfahrene und unerschrockene Benutzer eines solchen Systems, da sehr viel Information durch gute Visualisierungen auf einen Blick vermittelt werden kann. Speziell die Möglichkeit der Gruppierung von Information trägt zur Verständlichkeit der Oberfläche bei. Die Performanz ist mit dieser Art von Schnittstelle deutlich höher, und die Einarbeitungszeit für neue Benutzer ist deutlich geringer im Vergleich zur vorangegangenen Eingabeaufforderung, sofern die Anzahl der auf dem Bildschirm dargestellten Elemente den Benutzer nicht überfordert (siehe Abbildung 25). Je mehr Elemente dargestellt werden, umso weniger unterscheiden sich diese, was gegen die Forderung der Determiniertheit von Benutzungselementen spricht. Zudem kann nicht von einem Gefühl der Immersion gesprochen werden.

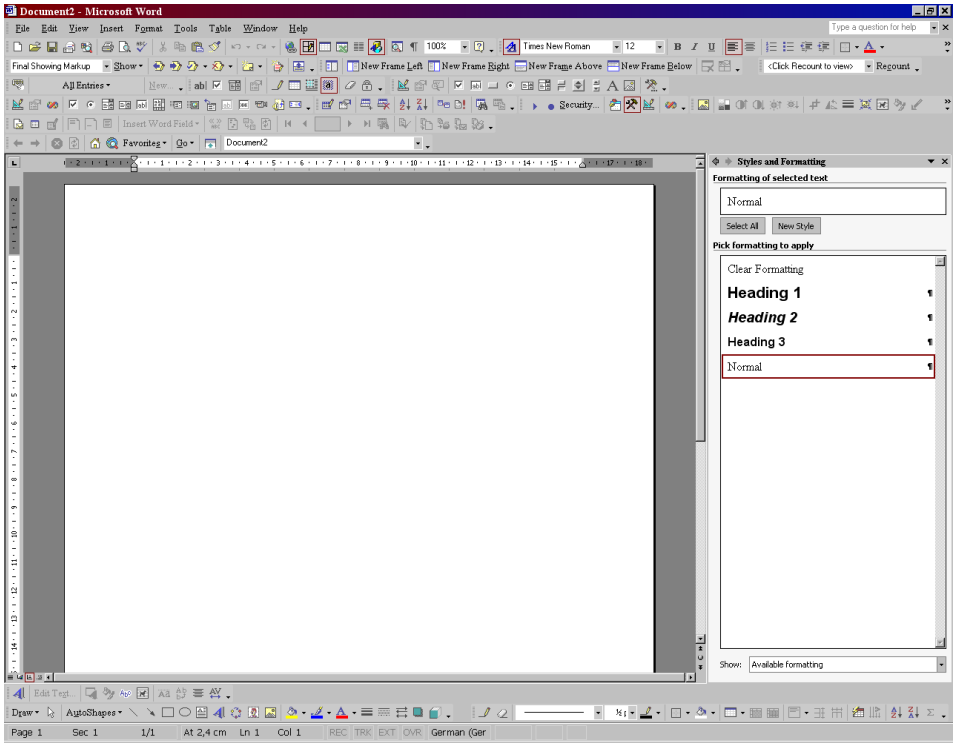


Abbildung 25 Benutzungsschnittstelle mit zu vielen Icons

Für die vorliegende Arbeit sind 2-D-Mensch-Maschine-Schnittstellen als Grundlage für die weiteren Entwicklungen von 3-D-Schnittstellen essentiell. Im Folgenden sollen nun die entsprechenden Grundlagen für den dreidimensionalen Fall eingeführt werden.

2.4.1 3-D-Interaktion

Dreidimensionale Benutzungsschnittstellen stellen gegenüber 2-D-Benutzungsschnittstellen eine erhöhte Schwierigkeit dar. In [72] werden 3-D-Interfaces als im gleichen Raum existierend wie die zu steuernde Anwendung bezeichnet. Außerdem werden die für 3-D-Interfaces notwendigen Vorraussetzungen aufgezeigt. Fast alle Anwendungen aus den konventionellen Einsatzgebieten von 3-D-Simulationen (CAD/CAM, Animationen, Medizin, Spiele, Visualisierung wissenschaftlicher Daten) haben bestimmte Formen der Interaktion gemeinsam:

- Skalierung: Viele wissenschaftliche Anwendungen beschränken sich in der Darstellung der Szene auf einen Teil eines speziellen Problems. Die Herausforderung ist hier, dass die entsprechenden Anwendungen jedoch in der Lage sein müssen, eine große Menge von Daten für den Benutzer interaktiv zugänglich zu machen.
- Level of Detail: Dem Anwender muss die deutliche Wahrnehmung relevanter Details ermöglicht werden. Dem gegenüber steht die technische Randbedingung, dass sich ein hoher Detailgrad negativ auf die Performanz der Visualisierungsanwendung auswirkt. „Level-of-Detail“-Techniken gestalten die Darstellungsgenauigkeit variabel, so dass ein Ausgleich zwischen den Anforderungen des Anwenders und der Performanz des Systems stattfinden kann. So ist es etwa entbehrlich, die Darstellung eines Objektes in

hoher Genauigkeit (hohe Anforderung an die Performanz des Systems) zu berechnen, wenn dieses Objekt in der Präsentation im Gesichtsfeld des Benutzers nur einen verschwindenden Platz einnimmt und z. B. nur durch wenige Pixel repräsenstiert wird.

- Interaktions-Modell: WIMP-Interfaces sind für die meisten 3-D-Anwendungen unpassend, besonders dann, wenn Ein- und Ausgabegeräte mit mehr als zwei Freiheitsgraden benutzt werden. Da die Kluft zwischen Anwendung und Interface deutlich geringer ausfällt als im 2-D-Fall, werden auch folgende Punkte wichtig(er):
    1. Eine höhere Bandbreite muss für Ein- und Ausgabe gewährleistet werden.
    2. Die Anzahl der Freiheitsgrade muss gesteigert werden.
    3. Es darf keine Kluft geben zwischen den Aktionen des Benutzers und der Reaktion des Systems; des Weiteren müssen die Antworten des Systems kontinuierlich erfolgen.
    4. Statt diskreter Eingaben muss eine kontinuierliche Bedienung des Systems ermöglicht werden.
    5. Bei Mehrbenutzersystemen müssen simultane Ein- und Ausgabekanäle unterstützt werden.
- Interaktionstechniken: Die konkreten Anforderungen der Anwendung bestimmen das Design der Benutzungsschnittstelle. Eine Schnittstelle zur Steuerung einer Architekturanwendung muss andere Techniken zur Verfügung stellen als eine Rennsimulation. Einige Schnittstellen zur Interaktion scheinen jedoch fast alle Anwendungen gemeinsam zu haben:
    1. Objekt-Erzeugung
    2. Objekt-Selektion
    3. Objekt-Platzierung und Manipulation
    4. Kontrolle der Perspektive
    5. Bereitstellung von Informationen über die Umgebung
    6. Spezifikation von Beziehungen von Objekten untereinander.

Abbildung 26 VR-Schnittstelle einer Anwendung

Bisher existiert keine umfassende, generisch einsetzbare Sammlung von Interaktions-Komponenten für 3-D-Anwendungen, obgleich einige grundlegende Techniken zu identifizieren sind. Beispielsweise sei hier das Ändern der Kameraposition genannt. Sollen nun derartige 3-D-Techniken für „klassische“ 2-D-Eingabegeräte, wie etwa die Maus, realisiert werden, ergeben sich Mehrdeutigkeiten. Eingaben des Benutzers müssen mit herkömmlicher Hardware erst umgesetzt werden. Die Hand bewegt die Maus, der Zeiger steuert ein Widget, die Interaktion mit dem Widget manipuliert das entsprechende Objekt. Eine Direktmanipulation mit Hilfe entsprechender Hardware mit vielen Freiheitsgraden kann diese Art von Problemen umgehen. Manchmal kann jedoch auch eine Beschränkung des Benutzers dessen Aktionen in eine sinnvolle Richtung leiten. Zum Zeichnen paralleler Linien ist der Einsatz eines Rasters beispielsweise eine etablierte Technik.

Nicht nur die Dateneingabe mittels 2-D-Technologien erzeugt Probleme, auch die Ausgabe dreidimensionaler Szenen auf konventionellen 2-D-Ausgabegeräten ist problematisch. Das grundlegende Problem ist hier die eindeutige Navigation in der virtuellen Welt; die Orientierung geht ohne entsprechende Hilfestellungen, wie etwa Orientierungspunkte oder -gitter, leicht verloren. Der Einsatz von Orientierungspunkten ist daher empfehlenswert. Entsprechend einfach muss auch das Interface zur Steuerung der Kameraposition gestaltet sein, da die beste Orientierungshilfe nutzlos ist, wenn der Benutzer nicht in der Lage ist, an den gewünschten Punkt zu gelangen.

Viele 3-D-Interfaces werden in Form von Metaphern beschrieben, deren Aussehen und Form die gekapselte Funktionalität visualisieren sollen.

Definition **Metapher [70]**:

„Essential concepts conveyed through words and images or through acoustic or tactile means. Metaphors concern both overarching concepts that characterize interaction and individual items, such as the "trashcan" standing for "deletion" within the "desktop" metaphor."

Die verschiedenen Eingabemetaphern lassen sich in zwei Klassen einteilen:

- physikalische Metaphern: aus der realen Welt entliehen
- unrealistische Metaphern: nicht in der realen Welt möglich

Metaphern, die aus der realen Welt entlehnt sind, richten sich oft nach der Zielgruppe, für die sie erstellt wurden. Eine „klassische" Metapher ist die „Desktop"-Metapher, die den meisten heute gängigen allgemeinen Benutzungsschnittstellen zugrunde liegt: Hier steht ein *Papierkorb* etwa für das Löschen von Dateien; *Ordner* stehen für Sammlungen von *Dokumenten*.

Aktions-Metaphern bilden eine Unterkategorie der Metaphern aus der realen Welt, sie beschreiben eine Bewegung oder eine Form der Manipulation, Drehen eines Rades, Drücken eines Knopfes, Auswählen oder Greifen.

Ein Beispiel für unrealistische Metaphern ist etwa das „Teleportieren" oder „Flug durch Wände einer 3-D-Szene".

2.4.2 Psychologie der virtuellen Realität

Für die Gestaltung von 3-D-Schnittstellen sind einige Besonderheiten der menschlichen Wahrnehmung zu berücksichtigen [12].

2.4.2.1 Navigation

Um im Raum zu navigieren, muss Wissen über die Umgebung, in welcher man sich bewegen möchte, vorhanden sein. Ob es sich dabei um die reale oder virtuelle Welt handelt, macht keinen Unterschied. Während einer Bewegung werden Informationen über Position, Geschwindigkeit und Beschleunigung ausgewertet. Positions- und Geschwindigkeitsindikatoren werden in der Regel durch das visuelle System ausgewertet. Es gibt allerdings Situationen, in denen keine Aussagen über die Geschwindigkeit möglich sind, so etwa sind in einem Flugzeug in großer Höhe Aussagen über Geschwindigkeit oder Beschleunigung für einen Menschen fast unmöglich. In einer virtuellen Umgebung sind Geschwindigkeitsinformationen jedoch leicht mit Hilfe geeigneter Visualisierungen vermittelbar. Anders sieht es bei der Simulation von Beschleunigungen aus. Es existieren zwar für einige Anwendungen geeignete Systeme für die Simulation von Beschleunigungskräften auf den Körper (Flugsimulator, manche Freizeitparks bieten entsprechende Fahrsimulationen an), allerdings sind diese sehr teuer und kaum flexibel. Eine „normale" 3-D-Simulation muss ohne solche Vorrichtungen auskommen. Die Auswirkungen sind, dass eine am Bildschirm dargestellte Bewegung nicht als solche wahrgenommen wird, vielmehr scheint sich die simulierte Szene um den Betrachter zu bewegen. Der Einsatz von Landmarken und anderen Orientierungshilfen ist in jedem Fall nötig, um dem Benutzer kontinuierlich Aussagen über die eigene Position zu ermöglichen.

2.4.2.2 Merkmale virtueller Objekte

3-D-Benutzungsoberflächen sollten unter Berücksichtigung folgender Richtlinien konzipiert werden [71]:

- Funktionelle Naturtreue (functional fidelity): Der Entwurf muss einem realen Gegenstück **nicht** fotorealistisch entsprechen. Das menschliche Gehirn ist in der Lage, die relevanten Merkmale eines Objekts zu extrahieren und sich darauf zu beschränken. Daher ist es ausreichend, nur die für eine bestimmte Aufgabe relevanten Merkmale eines Objekts darzustellen. Abbildung 27 verdeutlicht dies: Beide Objekte sind als Schachteln zu erkennen; auch die gezeichnete ist völlig ausreichend, um in ihr andere Objekte in einer Simulation zu verstauen. Man darf jedoch nicht vergessen, dass eine realistische Darstellung dem Gefühl der Präsenz in der Szene zuträglich ist. Die Darstellung einer quaderförmigen Kiste mit einer Holztextur wird aber das gleiche Ergebnis erzielen wie ein modellierter Weidenkorb.

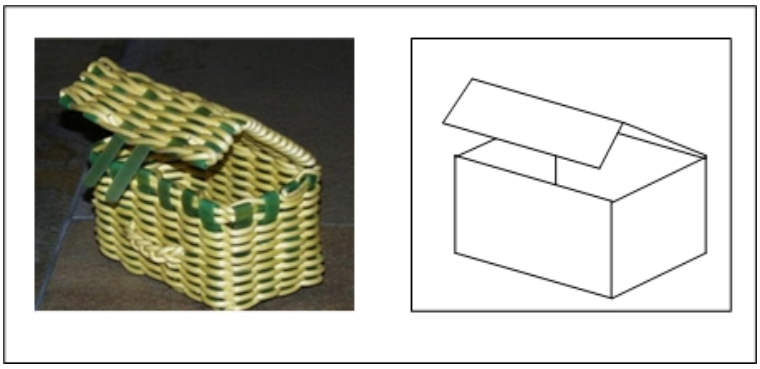


Abbildung 27 Abstraktion

- Reaktionsfähigkeit: Jede Aktion des Benutzers muss sofort vom System beantwortet werden. Nur so kann ein natürlicher Umgang für den Benutzer gewährleistet werden, denn jede Verzögerung bedeutet für den Benutzer erhöhte Konzentration, die Simulation ist nicht mehr interaktiv. Daher sollte größter Wert auf die Synchronisation aller Komponenten der Ein-Ausgabe-Kette gelegt werden. Es sollte auch Feedback für das Auslösen einer abstrakten Aktion gegeben werden, so dass der Benutzer immer darüber informiert ist, ob die Anwendung noch läuft oder die Berechnungen bereits beendet sind.
- Affordances: Schon das Erscheinungsbild der Schnittstelle sollte dem Benutzer deren Funktionalität verdeutlichen. Ein Schalter, welcher wie eine Lampe modelliert ist, lässt vermuten, dass das Licht damit geschaltet werden kann. Ein rundes Objekt verleitet eher zum Drehen als ein eckiges. Auch die verwendeten Materialien tragen entscheidend zur Auswahl des Verhaltens bei. Ein Gegenstand aus Glas wird vorsichtiger behandelt als einer aus Metall, eine glatte Oberfläche bietet keinen guten Halt, Papier eignet sich zum Schreiben.
- Erzeugen mentaler Repräsentationen (Appeal-to-Mental-Projections): Eine Schnittstelle muss auch als solche erkennbar sein.
- Mehrkanalige Ein- und Ausgabe: Benutzungsoberflächen sollten nicht nur den visuellen Kanal zur Kommunikation benutzen, es sollten so viele Sinne wie möglich stimuliert werden.

2.4.2.3 3-D-Interaktionen

Jeder Mensch ist ständig geistig und körperlich aktiv. Daraus folgt, dass auch die Interaktion mit der Schnittstelle einer Maschine ein kontinuierlicher Prozess ist, wenn auch nicht jede wahrgenommene Information zu einer Aktion des Benutzers führt. Zu nennen sind in diesem Zusammenhang Dinge wie Gravitation oder Reibung beim Manipulieren eines Objekts. Diese Informationen werden nicht bewusst verarbeitet, sie unterstützen jedoch in hohem Maß die Interaktion mit der Welt, sei sie nun real oder virtuell.

2.4.2.4 Ein- und Ausgabegeräte

Die Anzahl der Freiheitsgrade der Eingabegeräte muss den Anforderungen der jeweiligen Aufgabe angepasst sein. Sollte das Eingabegerät weniger Freiheitsgrade unterstützen, als die Aufgabe erfordert, also der Task Space ([72]) zu groß sein, sind wieder Metaphern zur Vereinfachung nötig, da direkte Manipulation nicht mehr möglich ist.

Auf der anderen Seite rechtfertigt diese Aussage absolut die Verwendung der Maus im 2D. Problematisch kann jedoch auch die Verwendung von Eingabegeräten mit vielen Freiheitsgraden bei Aufgaben mit kleinerem Task Space sein. Wie schon erwähnt, ist es nicht möglich, eine exakte Linie mit der Hand zu zeichnen. Hier sind wiederum Hilfen in Form von Metaphern nötig, in diesem speziellen Fall wäre dies zum Beispiel ein Lineal.

2.4.2.5 Benutzungsoberflächen, Design und der Benutzer

Ähnlich wie sich das Design von Eingabegeräten an der realen Welt orientiert, finden sich auch für die virtuellen Benutzungsoberflächen Metaphern aus der wirklichen Welt. Die bisher im Bereich des Schnittstellen-Designs gewonnenen Erkenntnisse könnten direkt von Entwicklern für Simulationen adaptiert werden, insbesondere bei der Gestaltung der Form und „Handlichkeit“ der Objekte. Das zu erreichende Ziel ist die Gestaltung von Schnittstellen, für deren Benutzung keine Beschreibung und idealerweise noch nicht einmal eine Beschriftung notwendig ist. Sicherlich wird dies in den meisten Fällen nur schwer zu erreichen sein. Es existieren jedoch Techniken für den sinnvollen Einsatz von Formen, Farben und Ikonen, womit deutlich mehr Informationen codiert werden können als durch den Einsatz von Text allein [63].

Die verschiedenen Einsatzbereiche von Benutzungsschnittstellen verlangen auch eine Anpassung des Designs an die jeweilige Benutzergruppe [69]. Des Weiteren muss berücksichtigt werden, dass eine solche Schnittstelle von Anfängern und Experten einer Benutzergruppe benutzt werden wird, entsprechend vielseitig und doch schlicht und deterministisch muss sie gestaltet werden. Der goldene Weg, also eine für alle Anwendungsfälle optimale Lösung, scheint demnach nicht zu existieren. Die Grundlagen für ein erfolgreiches Design, zumindest aus technischer Sicht, werden im Folgenden dargelegt.

2.4.2.6 Das konzeptionelle Modell

Von jedem Objekt, das wir wahrnehmen, erstellen wir ein mentales Modell, das durch unser Wissen und durch unsere Erfahrung geprägt wird. Dadurch sind wir in der Lage, Dinge in unsere Sicht der Welt einzuordnen, sie zu begreifen und ihnen Eigenschaften zuzuweisen, sie zu benutzen. Ist die Wahrnehmung unvollständig, so werden fehlende Teile durch gespeichertes Wissen ergänzt, was die mentale Repräsentation des Objekts zu einer individuellen Erfahrung macht. Gleiches gilt für mentale Repräsentationen semantischer Zusammenhänge wie Handlungsabläufe, welche für die Steuerung eines Geräts erforderlich sein können. Unterscheidet sich die Benutzungsschnittstelle eines Geräts nicht von der eines anderen mit

unterschiedlicher Funktionalität, ist eine Fehlbedienung so gut wie sicher, da der Mensch erlerntes Verhalten generalisiert, er ist ein „Gewohnheitstier“ [72].

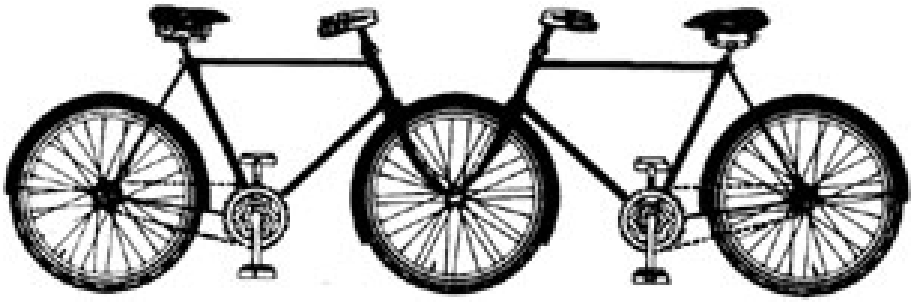


Abbildung 28 Carelmans Tandem, aus [12]

Das Tandem aus Abbildung 28 ist ein sehr gutes Beispiel für diese Fähigkeit. Kaum jemand wird versuchen, auf diesem Gefährt eine Probefahrt zu machen. Man weiß, dass zwei Lenker in der Mitte keinen Sinn ergeben, dass wir nicht in entgegengesetzte Richtungen fahren können, dass wir das Gleichgewicht nicht werden halten können. Dies ist jedoch nur möglich, weil wir die einzelnen Komponenten des Objekts deutlich sehen können, die Affordances sind in ausreichendem Maß wahrnehmbar. Das Design einer Schnittstelle unterliegt den gleichen Anforderungen: Es sollte dem Benutzer die Möglichkeit geben, ein konzeptionelles Modell des Geräts oder der steuerbaren Funktionen zu erstellen.

Das Fehlen eines konzeptionellen mentalen Modells führt zwangsläufig zu Fehlbedienungen, bedingt durch fehlendes Verständnis für die Funktionsweise des zu bedienenden Objekts. Man denke an das Konzept der Black Box, wie es bei vielen elektronischen Geräten zwangsläufig zum Einsatz kommt: Die Arbeitsweise solcher Geräte ist für Nicht-Techniker oft nicht nachvollziehbar. Der normale Benutzer sieht sich einem Kasten gegenüber, welcher auf „magische“ Weise die gewünschten Aufgaben ausführt. Wie dies im Einzelnen geschieht, kann nicht durch Exploration der Form (haptisch oder visuell) ergründet werden. Steht nur die schriftliche Bedienungsanleitung zur Verfügung, wird das konzeptionelle Modell allein durch diese Informationen erstellt, was oftmals die Verwirrung vervollständigt. Stimmt dann die Beschreibung nicht mit der tatsächlichen Implementierung überein oder wird nicht die gesamte Beschreibung gelesen, kann die Bedienung nicht mehr korrekt erfolgen. Das größte Problem ist jedoch, dass ohne ein generelles Verständnis für ein Objekt dessen Benutzung nicht generalisiert werden kann, da für nicht triviale Situationen kein Lösungsweg aus der mentalen Repräsentation des Objekts abgeleitet werden kann.

Das mentale Modell steuert dabei auch die Bewegungsabläufe während der Benutzung von Geräten. Die Computertastatur besitzt knapp über 50 verschiedene Tasten, welche ständig für Eingaben verwendet werden. Ein Anfänger ist ca. 100-mal langsamer als ein erfahrener Benutzer – unter der Voraussetzung, dass der Anfänger eine bis drei Sekunden pro Buchstabe benötigt und der Weltrekord bei über 500 fehlerfreien Anschlägen pro Minute liegt. Diese Leistung ist nur mit Hilfe des mentalen Modells erreichbar, welches zielgerichtete intuitive Bewegungen der Finger ermöglicht. Taktiles Feedback unterstützt die Bildung des mentalen Modells, indem Informationen über die Soll-Stellung der Finger für das Drücken einer bestimmten Taste geliefert werden. Dieses gespeicherte oder auch erinnerbare Konzept besteht demnach nicht nur aus visuellen Informationen, sondern genau aus den für die Benutzung und das Verständnis der Funktionsweise relevanten Informationen. Äußerliche Merkmale, wie die

Höhe und Breite, die Farbe oder das Gewicht der Tastatur, spielen in diesem speziellen Fall keine Rolle. Die Integration in unser Gedächtnis geht so weit, dass erfahrene „Tipper“ auf Kommando nicht sagen können, an welcher Stelle sich die Buchstaben befinden, beim Schreiben jedoch häufig benutzte Worte und Zeichenfolgen mit einer hohen Zahl von Anschlägen schreiben können.

2.4.2.7 Mapping

Im Sinne des Designs von Benutzungsschnittstellen beschreibt der Begriff Mapping die Beziehung zwischen den Bewegungen des Kontrollgeräts und deren Auswirkungen. Sinnvollerweise entspricht die Bewegung des Steuergeräts der Bewegung oder Aufgabe des zu steuernden Objekts. Eine seitliche Bewegung eines Objekts ist demnach durch eine seitliche Bewegung einer Kontrolleinheit am sinnvollsten umzusetzen. Bei räumlichen Bewegungen ist dies noch relativ einfach realisierbar. Problematisch wird es, sobald entweder das gesteuerte Objekt nur wenige Zustände hat, wie eine Lampe, oder die auszuführende Aufgabe von abstrakter Natur ist, so dass es keine natürliche Steuerung dafür geben kann. Eine Reihe von Lichtschaltern lässt normalerweise keinen Rückschluss darauf zu, welche Lampe mit welchem Schalter gesteuert wird. Ein Taschenrechner kann Zahlen addieren, eine Aufgabe, für die es keine natürliche Steuerung gibt, da es für sie keine Analogie in der realen Welt geben kann. Lichtschalter hingegen können so angeordnet werden, dass ihre Platzierung auf einer Montageplatte relativ zur Position der Lampen im Raum ist. Diese Montageplatte kann zusätzlich noch horizontal angebracht werden. So können Assoziationen zwischen der Steuerung und dem zugrunde liegenden Objekt geschaffen werden, welche die Bedienbarkeit vereinfachen, da ein mentales Bild geschaffen werden kann und auch ein Mapping zwischen Steuerung und Gerät möglich ist. Dies basiert allerdings auf der logischen Einsicht, dass die Anordnung der Schalter nicht willkürlich geschehen ist. Die möglichen Hilfestellungen durch eine sinnvolle Repräsentation der Benutzungsschnittstelle sind mit Zunahme des Abstraktionsgrades der auszuführenden Aufgabe von Erfahrung und Wissen des Anwenders abhängig.

Ein weiterer Aspekt ist die Anzahl der Kontrollmöglichkeiten. Man kann Tasten mit mehreren Funktionen belegen, abhängig vom aktuellen Zustand des Geräts. Diese Tasten können zusätzlich die Dauer des Drückens registrieren und entsprechend Aktionen auslösen. Auf der anderen Seite kann man jeder Funktion eine eigene Taste zuordnen. Diese könnte durch ihr Erscheinungsbild Aufschluss über ihre Funktion geben. Leider ist durch den Trend zur Miniaturisierung auch der Platz für die Kontrollelemente knapp geworden, was die Bedienung in den meisten Fällen erschwert, da wenige Tasten kontextabhängig mit vielen Funktionen belegt sind. Ein natürliches Mapping ist hier nicht mehr möglich.

2.4.2.8 Feedback

Da elektronische Geräte nicht bei ihrer Arbeit beobachtet werden können, fällt es Benutzern oft schwer, sich ein mentales Bild über den inneren Zustand des Geräts zu erzeugen. Durch gezieltes Feedback auf Aktionen des Benutzers kann jedoch der aktuelle Zustand des Geräts vermittelt werden. Dies kann ein simpler Ton beim Drücken einer Taste sein oder gar die permanente (und immer aktuelle) Anzeige des Zustands per Grafik-Display. Der Zustand des Akkus, Signalgüte, Datum und die aktuelle Belegung der oberen Tastenreihe einer Tastatur sind immer zu erkennen. Aktionen des Benutzers lösen in fast allen Fällen visuelles Feedback aus. Zusätzlich kann jeder Tastendruck mit einem Ton quittiert werden. Im Übrigen sind Handys gute Beispiele für schlechtes Mapping, da jede Taste mit mehreren Funktionen belegt ist. Wer schon den Short-Message-Service (SMS) benutzt hat, konnte feststellen, wie ermüdend das Suchen der Buchstaben auf der Tastatur ist, so dass sich die Zeit für das Schreiben eines Texts

um einen Faktor größer als zwei erhöht. Zum einen müssen die Tasten mehrfach gedrückt werden, bis der gesuchte Buchstabe erscheint, zum anderen werden SMS nur einhändig, sehr oft auch nur mit einem Finger geschrieben (verglichen mit einer normalen Tastatur).

2.4.2.9 Human Action Cycle

Menschliches Verhalten kann auf verschiedene Weisen im Modell angenähert werden. Einige Ansätze gehen von der Vorstellung aus, dass unser Gehirn in einer Art Informationsverarbeitungskreislauf arbeitet. In [73] wird das Modell des Human Action Cycle definiert. Demnach werden sensorische Wahrnehmungen nach dem Empfang interpretiert, was wie der gesamte kognitive Prozess von den Erfahrungen und dem Wissen des Menschen abhängig ist. Die Interpretation der Welt zum aktuellen Zeitpunkt wird bewertet, wobei objektiv gleiche Situationen von verschiedenen Menschen völlig unterschiedlich bewertet werden können [77]. Aus der subjektiven Sicht der Dinge werden neue Ziele (goals) formuliert, für deren Erreichen Aktionen notwendig sind, welche ausgeführt werden und den Einfluss des Menschen auf die Welt verkörpern (Abbildung 29).

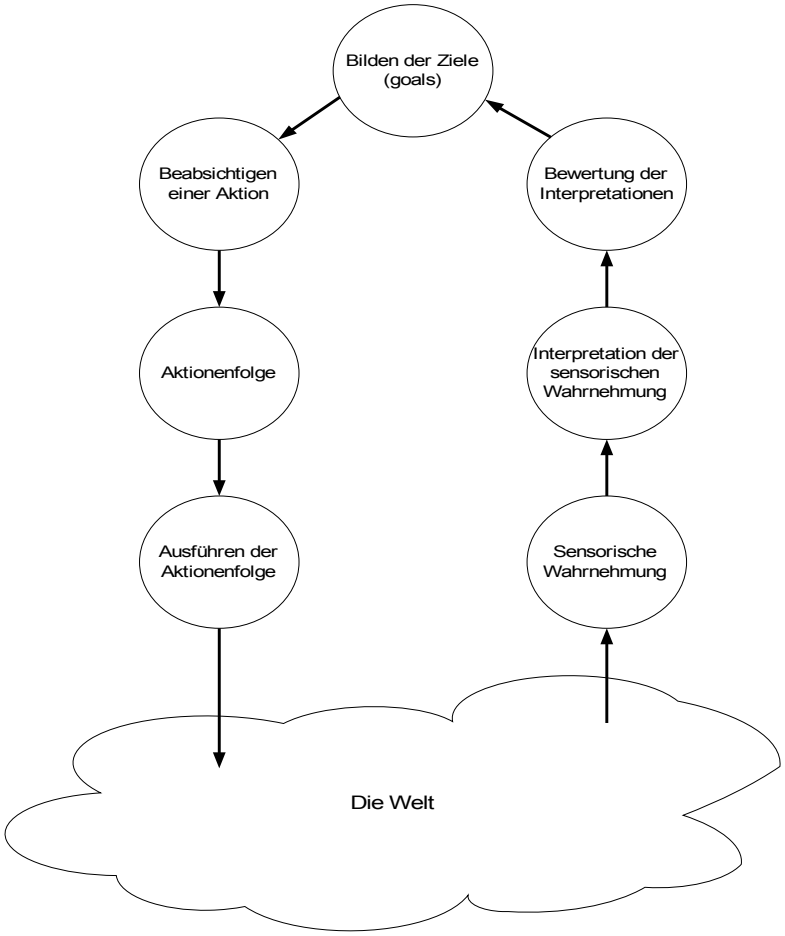


Abbildung 29 Human Action Cycle

Der Ablauf ist jedoch nicht als strikt sequentiell anzusehen. Es ist kein fester Zeitpunkt für die einzelnen Schritte des Modells nachweisbar, auch beschreibt das Modell nicht vollständig den Prozess der Kognition. Es ist vielmehr ein Prozess einer kontinuierlichen

Informationsaufnahme und Verarbeitung, welcher im Ganzen als Kreislauf ohne zeitlichen Rahmen angesehen werden kann.

2.4.2.10Beschränken der möglichen Alternativen: Constraints

Durch Beschränkung der möglichen Alternativen bei der Bedienung eines Geräts kann die kognitive Last deutlich reduziert werden, da kein oder nur ein minimaler Entscheidungsprozess durchlaufen werden muss. Durch die Auswahl und Bereitstellung von Formen und Materialien mit der Aufgabe entsprechenden Affordances und Beschränkungen (Constraints) kann der Benutzer jedoch entsprechend entlastet werden.

Abbildung 30 „Babys erste Bausteine“ von Fisher-Price

Für den Bereich des Designs von virtuellen Objekteigenschaften kann man vier Klassen von Constraints unterscheiden:

- Physikalisch: Diese Constraints basieren nur auf der wahrgenommenen Umgebung und sind nicht vom Wissen des Benutzers abhängig. Ein Gegenstand, welcher nicht getragen oder gehoben werden soll, kann „festgeschraubt“ sein, der Benutzer kann die Aktion „Heben“ nicht durchführen. Man trifft in seiner Umgebung ständig auf diese Art der Beschränkung: Der Schlüssel passt nur in eine Richtung in das Schlüsselloch, eine Taste lässt sich nur drücken. Abbildung 30 zeigt eine Beschränkung der möglichen Alternativen durch die Form: Das Runde passt nicht in das Eckige!
- Semantisch: Nicht jede mögliche Aktion ergibt in jeder Situation Sinn. Ein Bild wird meistens an die Wand gehängt und nicht an das Fenster, eine heiße Pfanne stellt man besser auf einen Untersetzer und nicht auf den lackierten Holztisch. In der realen Welt sind semantische Constraints vom Wissen des Menschen abhängig, in einer virtuellen Welt können auch Objekte mit Wissen über sich selbst ausgestattet werden, was den

Benutzer einer Virtual-Reality-Anwendung sehr gut unterstützen kann, da Fehlbedienungen in gewissem Maß vom Objekt vermieden oder verhindert werden können.

- Logisch: Ein Kreuzworträtsel hat für manche Fragen mehrere richtige Antworten. Ist in einer kreuzenden Folge bereits eine Antwort eingetragen, wird die Anzahl der möglichen Antworten für die Frage mit mehreren Lösungen logisch beschränkt. Logisch sind auch die oben genannten Zusammenhänge beim Mapping. Diese Beschränkungen sind in hohem Maß vom Wissen des Menschen abhängig, da Entscheidungen allein aus dem bereits erworbenen Wissen über die Welt getroffen werden.
- Kulturell: Wenn es trotz physikalischer, semantischer und logischer Beschränkungen mehr als eine sinnvolle Alternative gibt, können kulturelle Erfahrungen des Menschen die Entscheidung beeinflussen. In manchen Ländern Asiens steht die Farbe Rot zum Beispiel für Feierlichkeiten und wird bei Hochzeiten etc. verwendet. Dagegen steht in Europa Rot für „Gefahr“ oder „Achtung!“. Ähnliches gilt auch für Menschen mit verschiedenen Berufen. So kann für einen Techniker die Farbe Grün bedeuten, dass alles in Ordnung ist, für einen Mediziner kann es ein Warnsignal sein.

2.4.3 Manipulation

„Ma|ni|pu|la|ti|on, die; -, -en (Hand-, Kunstgriff; Verfahren; *meist Plur.:* Machenschaft)“[188]

Manipulation wird hier entsprechend als „Handgriff“, „Verfahren“, „Interaktion mit den Händen“ verstanden. Im folgenden Kapitel sollen die besonderen Voraussetzungen und Anforderungen diskutiert werden, die die Interaktion mit den Händen und mit haptischem Feedback an computergraphische Anwendungen stellt.

2.4.3.1 Der Mensch hat zwei Hände

Die Forderung nach einem System, welches eine direkte Manipulation der virtuellen Objekte ermöglicht, bedeutet auch, dass der Einsatz der Hände des Benutzers auf natürliche Art unterstützt werden muss. Kommt eine virtuelle Hand zum Einsatz, ist dieses Problem sofort gelöst. Jedes andere Eingabegerät als der Datenhandschuh stellt eine Reduktion der Freiheitsgrade der Hand dar. Dass dies nicht immer schlechter ist, zeigt dieser Abschnitt.

Wenn man eine Benutzungsschnittstelle konzipieren möchte, welche mit den Händen bedient werden soll, ist natürlich auch eine Untersuchung der Möglichkeiten der Hand und der für sie ausführbaren Aufgaben nötig. Leider sind für den Bereich der Computertechnik nicht sehr viele Untersuchungen durchgeführt worden, so dass an dieser Stelle mehr Fragen offen bleiben als beantwortet werden. Möglicherweise liegt dies an schlechter oder fehlender Hardware, welche den Einsatz der Hände in Virtual-Reality-Anwendungen nicht sinnvoll erscheinen lässt, so dass Entwickler und Forscher ihre Arbeit in andere Projekte investierten. Vielleicht mangelt es auch an haptischem Feedback bei reinen Datenhandschuhen, was die Interaktion mit virtuellen Objekten genauso schwierig macht, wie es mit konventionellen Eingabegeräten der Fall ist.

Die Hände bieten in der realen Welt noch immer eine unerreichte Zahl von Einsatzmöglichkeiten, und es gibt keinen Grund, sich diese nicht auch in virtuellen Umgebungen zunutze zu machen. Der hohe Aufwand macht allerdings die Prüfung des zu erwartenden Vorteils notwendig. Dieser kann durch Einsparungen materieller Art, eine erhöhte Performanz beim Bearbeiten von Aufgaben oder eine geringere Einarbeitungszeit bei neuen Anwendungen erreicht werden.

Die für diese Arbeit wichtigsten Untersuchungen und Erkenntnisse, welche in Zusammenhang mit Interaktion in virtuellen Welten stehen, werden hier vorgestellt. Der aktuelle Wissensstand erlaubt leider keine Vorgabe einer einheitlichen Vorgehensweise bei der Planung von graphischen oder gar haptischen Benutzungsoberflächen, daher sind die folgenden Ausführungen als Richtlinie zu betrachten.

2.4.3.2 Ist der Einsatz der Hände sinnvoll?

Die Hände bieten ein deutlich breiteres Spektrum an Einsatzmöglichkeiten als jedes andere Eingabegerät. Zudem sind sie ein Teil unseres Körpers, was eine intuitive, nicht kognitiv belastende Benutzung erlaubt. Stellt man die Frage, ob wirklich jede Aufgabe durch den Einsatz der Hände als Eingabegerät beschleunigt oder verbessert werden kann, so lautet die Antwort klar: „Nein!“. Die vielleicht häufigste Aufgabe am Computer, das Eingeben von Worten, ist mit heutigen Mitteln, z. B. der Tasatatur, effizient durchführbar. Das Eingabegerät ist der Aufgabe gut angepasst, auch wenn es einiger Übung bedarf. Eine Umstellung auf den Einsatz virtueller Hände ist wenig sinnvoll, da eine ebenbürtige virtuelle Alternative für Texteingaben nicht bekannt ist. Ein anderes Beispiel ist das Zeichnen von Linien im 2D. Mit Hilfe der Maus und eines Rasters für die genaue Positionierung ist diese Aufgabe gut und schnell auszuführen.

Was wäre jedoch, wenn die Aufgabe komplizierter ist: Translation eines Objekts, verbunden mit einer Rotation um neunzig Grad, zusätzlich soll das Objekt noch verformt und die Oberfläche abgetastet werden. Mit dieser in der realen Welt ganz natürlichen Aktion überfordert man jedes bekannte Eingabegerät (allein durch das Abtasten der Oberfläche). Was aber macht diesen Unterschied aus? Den steigenden Anforderungen der Anwendungen in Bezug auf deren Steuerung sind konventionelle Eingabegeräte nicht gewachsen, da sie die gleichzeitige Ausführung der Manipulationen nicht ermöglichen. Vor allem im Bereich der 3-D-Simulationen wird der Einsatz von Eingabegeräten mit vielen Freiheitsgraden erforderlich, weil dem Benutzer ein möglichst einfacher Zugang gegeben werden soll.

2.4.3.3 Geste und Postur

Gesten bieten eine Möglichkeit zur Kommunikation ohne Sprache und Schrift. Jeder Ausdruck kann durch entsprechende Gesten deutlich gemacht werden. Die Gebärdensprache hörgeschädigter Menschen zeigt dies. Die folgenden Punkte beschreiben die Probleme beim Einsatz ([78], [79], [80], [81]):

- Gesten werden für die Ausführung von Aktionen eingesetzt, was eine natürliche (oder natürlichere) Art die Interaktion mit der Anwendung erlauben soll. Aktionen wie Navigation oder Selektion sind jedoch keineswegs durch die verwendeten Gesten auf natürliche Weise zu steuern. Die begrenzte Anzahl möglicher Gesten schränkt zusätzlich stark ein. Man denke an eine Zeige-Geste: Den Zeigefinger nach vorne zu strecken, ist für die meisten Menschen kein Problem. Den Mittelfinger wird man für eine solche Aufgabe nicht benutzen (erst recht nicht in Multi-User-Umgebungen). Den Ringfinger werden die wenigsten Menschen unabhängig von den anderen Fingern

strecken können, und den kleinen Finger setzt kaum jemand für diese Geste ein (es wäre keine natürliche Aktion).

- Die Erfassung von Gesten durch die Software stellt ein großes Problem dar. Die Frage ist, ab wann der Benutzer die Geste zeigt und nicht seine Hand für andere Aufgaben benutzt oder sie nicht der Spezifikation der Geste entsprechend hält, obwohl er der Meinung ist, dass er klare Eingaben produziert.
- Die Verfügbarkeit von Technologien zur Vermittlung von haptischem Feedback und die in naher Zukunft verfügbare Steuerung durch Sprache bieten einen leichteren und natürlicheren Zugang, wenngleich Gesten durchaus zum alltäglichen Repertoire menschlicher Kommunikation gehören. Die ausgedrückte Information ist meist nur eine ungenaue Untermalung gesprochener Worte.

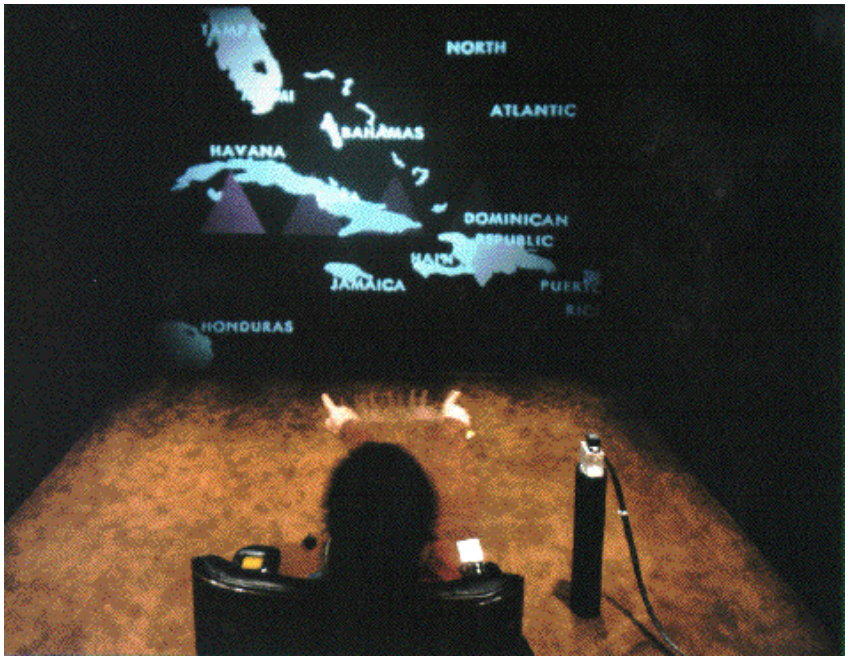


Abbildung 31 "Put-That-There"

Eine Ausnahme gibt es jedoch: Die Postur, welche einen echten Informationswert besitzt, ist die eben erwähnte Zeige-Postur. Sogar ein Finger wurde nach ihr benannt. Zudem ist sie die erste koordinierte Handbewegung des Menschen und wird wohl fast jeden Tag in Verbindung mit Sprachinformationen eingesetzt: „Put that there“ [82], (Abbildung 31).

2.4.3.4 Klassifizierung der Hand-Aktionen

Für genauere Betrachtungen ist ein Modell der möglichen Aktionen der Hand nötig, wobei zwei Hauptkategorien aufgestellt werden können:

- Kontinuierliche Aktionen: Sie basieren auf der Anzahl der Freiheitsgrade der Hand und beinhalten kontinuierliche Größen, wie Position der Fingerspitzen, Drehgeschwindigkeit der Gelenke oder einwirkende Kraft auf die Hand. Gesten fallen in diesen Bereich, aber auch das Bewegen der Maus oder des Joysticks.

- Diskrete Aktionen: Sie basieren auf vordefinierten Werten der Eigenschaften der Hand. Beispiele sind das Ballen zur Faust oder das Zeigen auf ein Objekt mit dem Zeigefinger oder das Drücken der Maustaste.

Beide Kategorien können auf drei verschiedene Arten interpretiert werden (direkt, abgebildet (mapped), symbolisch), so dass Hand-Aktionen insgesamt durch sechs Kategorien beschrieben werden [80]:

- Kontinuierlich/direkt: Die Daten einer virtuellen Hand werden kontinuierlich den realen, kinematischen Werten angepasst.
- Kontinuierlich/abgebildet: Die kontinuierlichen Daten der realen Hand werden auf ein Eingabegerät abgebildet. Das Bedienen der Computer-Maus ist eine Abbildung der Bewegungen der Hand.
- Kontinuierlich/symbolisch: Die Anwendung interpretiert die kontinuierlichen Hand-Daten und interpretiert die Intention des Benutzers. Eine Geste in einer virtuellen Umgebung fällt in diese Kategorie.
- Diskret/direkt: Diskrete Hand-Posturen werden direkt in eine manipulierende Aktion überführt.
- Diskret/abgebildet: Diskrete Hand-Posturen aktivieren eine diskrete Aktion. Beispielsweise bewegt sich die virtuelle Kamera in Richtung eines ausgestreckten Zeigefingers.
- Diskret/symbolisch: Natürliche diskrete Hand-Posturen werden benutzt, um Kommandos für eine Anwendung zu implementieren, wie das Ausführen einer Stop-Postur (Handrücken in Richtung Schulter, alle Finger nach oben gerichtet), um ein Objekt zu stoppen.

Demnach ist das Benutzen der Maus in einer Window-Umgebung eine Kombination aus kontinuierlichen und diskreten Aktionen. Diskrete Operationen finden sich allgemein im Bereich künstlich erzeugter Dinge, insbesondere im Bereich von Benutzungsschnittstellen. Kontinuierliche Operationen sind die natürliche Art der Benutzung der Hände. Man kann ganz allgemein sagen, dass der Mensch von Natur aus keine diskreten Abläufe kennt. Er hat sie erfunden, da sie die Kommunikation in einigen Bereichen vereinfachen, z. B. im Straßenverkehr oder generell im Bereich von Technik und Mathematik. Für die Informationsaufnahme hat diese Methode sicher ihre Berechtigung; eine diskrete Steuerung einer Maschine mit Hilfe von Tasten und Reglern ist bei zweckmäßiger Anordnung und entsprechendem Feedback oft sinnvoll. Die Steuerung eines Computerprogramms mit Maus und Maustaste bedeutet für den Benutzer jedoch eine hohe kognitive Belastung, da schon die Positionierung des Zeigers volle Aufmerksamkeit verlangt. Die Bewegung ist obiger Taxonomie entsprechend kontinuierlich, jedoch so schwer zu steuern, dass sie nicht intuitiv ausgeführt werden kann.

Durch den Einsatz virtueller Hände kann zumindest der Stand der Technik der realen Welt der Benutzungsschnittstellen auf die Software-Technologie angewendet werden. Wie schon erwähnt, ist besonders der Bereich der 3-D-Simulationen in diesem Zusammenhang von Bedeutung, mit deren Hilfe die Steuerung einzelner Objekte durch die Hände stark vereinfacht werden und eine direkte Manipulation erfolgen kann, sofern ein haptisches Feedback vom System geliefert wird.

Ohne den Einsatz von haptischem Feedback ist die Steuerung einer Anwendung durch virtuelle Hände nur mit Hilfe weiterer Eingabemetaphern möglich. Diese können aber auch durch

herkömmliche Eingabegeräte gesteuert werden. Möchte man beispielsweise ein Objekt greifen, wird der Moment der Berührung vor allem taktil wahrgenommen. Die erforderliche Hand-Postur ergibt sich aus der Form des Objekts. In einer virtuellen Umgebung ohne haptisches Feedback kann dieser Mechanismus nicht funktionieren, so dass der Zustand „Gegriffen" nicht bestimmt werden kann. Dieser Begriff soll nicht als diskreter Zustand verstanden werden, sondern umschreibt den kontinuierlichen Prozess des Anpassens der von der Hand aufzubringenden Kräfte, so dass das Objekt gehalten werden kann. Der Benutzer muss eine weitere Aktion zusätzlich zum Positionieren der Hand und der Finger ausführen, um der Anwendung zu signalisieren, dass er gegriffen hat. Die eigentlich abgeschlossene Aktion muss also zusätzlich bestätigt werden.

2.4.3.5 Bezugsrahmen und kinematische Ketten

Wir haben ein einfaches Modell der Einsatzmöglichkeiten der Hand. Es stellt sich jedoch die Frage, welche Einschränkungen es gibt und welche Vorraussetzungen erfüllt sein müssen, damit die Hand sinnvoll eingesetzt werden kann. Es existiert zurzeit kein Modell, welches eine genaue Klassifizierung zweihändiger Aktionen ermöglicht.

Die Hand ist Teil einer kinematischen Kette, eines Systems, welches aus starren, in Serie platzierten Teilen (Verbindungen) besteht, die durch Gelenke miteinander verbunden sind ([83], [82]). Für den menschlichen Arm bedeutet dies, dass der Oberarm mit der Schulter verbunden ist, der Unterarm mit dem Oberarm, die Hand mit dem Unterarm, die Finger mit der Hand. Jedes Teil der Kette wird in seinem Einflussbereich von den hierarchisch über ihm stehenden Teilen beeinflusst: Wird der Oberarm bewegt, hat dies Einfluss auf die Position des Unterarms, und dies wiederum hat Einfluss auf die Hand. Interessant ist, dass der Einsatzbereich der Hand durch diese Vorgaben im Wesentlichen auch auf das Gesichtsfeld beschränkt wird.

Die Hand ist jedoch nicht nur durch die Anatomie des Menschen beschränkt. Die beiden Hände erledigen unterschiedliche Aufgaben. Man unterscheidet hier zwischen der dominanten und der nicht-dominanten Hand: bei einem Rechtshänder ist die rechte Hand die dominante. Im Zusammenspiel scheint die nicht-dominante Hand der dominanten als eine Art „Wegbereiter" zu dienen, so dass die dominante bei der Ausführung einer Aufgabe unterstützt wird. Somit gibt die nicht-dominante Hand den Bezugsrahmen für die dominante vor. Das Schreiben auf ein Stück Papier verdeutlicht diesen Ablauf:

- Räumlicher Bezugsrahmen von dominant zu nicht-dominant: Die nicht-dominante Hand hält und positioniert das Blatt, welches von der dominanten Hand beschrieben wird.
- Ungleichgewicht der Bewegungen: Die dominante Hand wird deutlich häufiger und schneller eingesetzt als die nicht-dominante, die Hände arbeiten asymmetrisch. Das Blatt wird deutlich seltener durch die nicht-dominante Hand bewegt als der Stift durch die dominante.
- Vorrang der nicht-dominanten Hand: Die nicht-dominante Hand beginnt normalerweise den Ablauf des asymmetrischen Ablaufs. Das Blatt wird positioniert, dann erst beginnt die dominante Hand das Schreiben.

Dieses Modell gilt nur für die Ausführung von Aufgaben, bei denen auch beide Hände zum Einsatz kommen und die in Teilaufgaben unterteilt werden können. Das Fangen eines Tennisballs wird nur von einer Hand unabhängig von der anderen durchgeführt (normalerweise der dominanten). Das Fangen eines Fußballs erfordert den Einsatz beider Hände, hier sind

beide Hände in gleichem Maß beteiligt, die Aufgabe an sich ist jedoch nicht von abstrakter Natur.

Direkte Manipulation von Objekten in virtuellen Welten erfordert aber genau das oben beschriebene Zusammenspiel der Hände, so dass für den Bereich der Benutzungsschnittstellen davon ausgegangen werden kann, dass sich der Bezugsrahmen für die Interaktion mit virtuellen Objekten zwischen den Händen befindet, dynamisch bestimmt durch die Position der nicht-dominanten Hand. Die nicht-dominante Hand führt normalerweise intuitive Aufgaben aus, welche demnach nicht von der eigentlichen, abstrakten Aufgabe der dominanten Hand ablenken, und ist somit eine unersetzliche Hilfe, deren Fehlen sich auch bei scheinbar „einhändigen“ Aufgaben wie Schreiben sofort unangenehm und Performanz-reduzierend auswirkt.

2.4.3.6 Input-Performanz

Ob der Einsatz der Hände allgemein einen echten Performanzgewinn gewährt, ist ebenfalls kaum erforscht. Das Problem bei Aussagen über die Performanz besteht darin, dass bisher kein Modell existiert, dass Aussagen über die Lösbarkeit von Aufgaben mit Hilfe der Hand zulässt [84], [80], [83], [85]. Dies wird auch nur äußerst schwierig aufzustellen sein, da zum einen kontinuierliche Aktionen qualitativ bewertet werden müssten (wobei unterschiedliche Vorgehensweisen zum gleichen Ergebnis führen können), zum anderen wäre eine Untersuchung aller möglichen Kombinationen von Hand-Aktionen erforderlich, so dass ein Gesamtbild entstehen könnte. Die folgenden Aussagen zeigen einen Rahmen für die Fähigkeiten der Hand:

- Eine perfekte Linie kann nicht gezeichnet werden, sei es mit einer oder mit beiden Händen. Für solche Aufgaben funktioniert die Hand nicht genau genug. Das Zeichnen von Bildern jedoch kann von geübten Menschen fast foto-realistisch durchgeführt werden. Der Unterschied besteht darin, dass beim Zeichnen kontinuierliches visuelles Feedback dargeboten wird, so dass der Zeichner kontinuierlich die nächste Bewegung an den Gesamteindruck anpassen kann. Das Ziehen einer Linie bietet keinen Spielraum für Fehler.

- Der Einsatz von Werkzeugen erlaubt das Ausführen von Aufgaben, welche durch die „nackte“ Hand nicht durchführbar wären. So kann ein Nagel nicht ohne Hammer in die Wand geschlagen oder ein Faden nicht ohne Nadel in Stoff eingewebt werden. Eine perfekte Linie kann mit Hilfe eines Lineals gezeichnet werden. Die Hand kann in diesem Fall das Werkzeug auf durchaus unterschiedliche Arten benutzen, so dass es noch besser an die jeweilige Aufgabe angepasst werden kann. Auch können Werkzeuge für Aufgaben eingesetzt werden, für die sie nicht konzipiert wurden. So kann mit einem Messer eine Schraube gedreht oder mit einem Glas ein Teig ausgerollt werden.

- Manche Aufgaben lassen sich mit Hilfe der Hände oder mit Hilfe von Werkzeugen lösen. Die Wahl der passenden Methode hängt in diesem Fall von der Übung und den Fähigkeiten des Menschen ab.

Wenn man nun die Anforderungen an eine Benutzungsoberfläche betrachtet, kann man das Benutzen von Eingabegeräten durchaus als das Benutzen von Werkzeugen betrachten. Die Performanz und Benutzbarkeit muss dann im Einzelfall getestet werden. Wenn die Aufgabe nur zwei Freiheitsgrade erfordert, beispielsweise das Positionieren des Zeigers, so wird der Einsatz einer virtuellen Hand zur Direktmanipulation des Zeigers kaum einen Performanzvorteil bringen, da die vielen Freiheitsgrade der Hand nicht benötigt werden.

Aufgrund obiger Aussagen könnte man zu der Ansicht gelangen, dass die rein anatomisch gesehen identischen Hände des Menschen rein funktional gesehen völlig unterschiedlich sind.

Dies ist nur bedingt richtig. Der Mensch kann sich nur auf eine schwierige Aufgabe gleichzeitig konzentrieren. Schwierig bedeutet in diesem Zusammenhang, dass die volle Aufmerksamkeit auf diese Aufgabe gerichtet ist. Das Bearbeiten zweier schwieriger Aufgaben bedeutet ein geistiges Umschalten zwischen diesen, ein paralleler Ablauf ist nicht möglich. Ist nun diese Aufgabe durch den Einsatz einer einzelnen Hand durchführbar, so wird die dominante Hand eingesetzt. Die nicht-dominante Hand könnte diese Aufgaben bei Bedarf in vollem Umfang übernehmen, allerdings ohne die feinmotorische Abstimmung der dominanten Hand, was an mangelnder Übung der nicht-dominanten Hand liegt. Sogar die Füße können viele Aufgaben der Hand übernehmen, wenn sie entsprechend trainiert werden. Im Normalfall wird dennoch die dominante Hand die feinmotorischen Aufgaben übernehmen. Um einen Performanzgewinn zu erzielen und intuitive Bedienung zu gewährleisten, sollten dem Benutzer nicht zwei schwierige Aufgaben gleichzeitig zugemutet werden.

2.4.3.7 Haptik

Taktile Wahrnehmung, auch haptische Wahrnehmung genannt, ist die Voraussetzung für jede Art von koordinierter Bewegung. Sie liefert die nötige Rückmeldung über Position und Haltung unseres Körpers. Fehlen diese Informationen, sind besonders feinmotorische Aufgaben kaum noch zu erfüllen. Das Greifen eines Objekts mit einer virtuellen Hand wäre allein von der Qualität der räumlichen Darstellung und der räumlichen Wahrnehmung des Benutzers abhängig. Das Fehlen taktiler Informationen fällt auf und wirkt störend. Simulieren lässt sich dies bei Kälte, da das haptische System dann nicht mehr funktioniert [86].

Der visuelle Sinn ist zwar der dominante, es treten jedoch bei mangelnder Versorgung mit verwertbarer Information die anderen Sinne immer weiter in den Vordergrund. Man denke an einen schlecht beleuchteten Raum, in dem man durch Tasten den Weg zur Tür oder zum Lichtschalter sucht. Dieses Beispiel veranschaulicht auch die Mächtigkeit dieses Sinns: Allein durch aktives Fühlen kann ein kognitives Bild der gesamten Umgebung erstellt werden, inklusive Informationen über die Oberflächenbeschaffenheit von Objekten [87]. Allerdings ist die verfügbare Auflösung begrenzt: An den Stellen mit der höchsten Auflösung, den Fingerspitzen, müssen zwei Objekte mindestens 2,5 Millimeter voneinander entfernt liegen, um nicht als nur ein Objekt gefühlt zu werden. Die Höhe eines Objekts relativ zum Untergrund braucht hingegen nur wenige tausendstel Millimeter zu betragen, um wahrgenommen zu werden. In [87] wird sogar ein Weg aufgezeigt, wie aus graphischen Daten in Echtzeit taktile Information für ein dynamisches taktiles Display gewonnen werden können.

Der Einsatz von Haptik in virtuellen Realitäten ist zudem eine Möglichkeit, die Bandbreite der vermittelbaren Informationen zu erhöhen, ohne kognitive Ressourcen zu verbrauchen. Durch die Zugabe von haptischem Feedback wird auch die Direktmanipulation von Objekten mit Hilfe von virtuellen Händen ermöglicht. Ohne Feedback ist es nicht möglich, ein Objekt zu greifen oder zu drehen, da die Form nicht erfühlt werden kann, man greift hindurch. Dies führt zu einem Widerspruch zwischen in der realen Welt Erlerntem und der Wahrnehmung in der Simulation. Die Bedienung wird unnatürlich, und es entsteht der Bedarf an Eingabemetaphern, was den potentiellen Vorteil der virtuellen Hand als Eingabegerät fraglich werden ließ.

2.4.3.8 Passive haptische Wahrnehmung

Haptisches Feedback ist wie sensorische Wahrnehmung im Allgemeinen ein kontinuierlicher Prozess. Wir nehmen viele Dinge nur unbewusst wahr und werten die gewonnenen Informationen nicht aus, solange sie nicht unsere Aufmerksamkeit erregen. So fühlen wir die Temperatur, Oberflächen, Formen oder auch auf die Hand einwirkende Kräfte beim Händeschütteln. Weicht unsere Wahrnehmung jedoch von gewöhnlichen Mustern ab, wird sich

die Konzentration auf diese Abweichung lenken. Passives Feedback hilft uns, außergewöhnliche Zustände zu erkennen. Die Wahrnehmung von Schmerz dient gar unserer Sicherheit. Dies macht einen nicht offensichtlichen Aspekt der Wahrnehmung deutlich: Wenn wir denken, dass wir keine Schmerzen haben, sind die von den Nerven transportierten Signale nicht stark genug, um Schmerz empfinden zu lassen. Ein Signal wird dennoch übermittelt.

Abbildung 32 Haptische Wahrnehmung ist nicht auf die Hände beschränkt.

2.4.3.9 Oberflächenbeschaffenheit

Oberflächenstrukturen können durch Darbietung von Vibrationen an den Fingerspitzen vermittelt werden, da die Sensoren in der Haut des Menschen durch Streichen über eine Oberfläche in Schwingungen versetzt werden. Dieser Mechanismus kann durch künstliche Vibrationen getäuscht werden [86]. Diese Art der Stimulation erlaubt ebenfalls ein realistischeres Erleben der virtuellen Umgebung.

2.4.3.10Aktive haptische Wahrnehmung

Zusätzlich zu den passiv erworbenen Informationen kann haptische Information auch bewusst aktiv erworben werden. In [88] werden folgende Arten der haptischen Exploration unterschieden:

- Seitliche Bewegung: Beurteilung der Oberflächenbeschaffenheit
- Ausüben von Druck: Beurteilung der Härte, normalerweise im rechten Winkel zur Oberfläche
- Statischer Kontakt: Beurteilen der Temperatur
- Freier Halt: Beurteilung des Gewichts
- Umgreifen: Bestimmung von globaler Form und Volumen
- Verfolgen von Konturen: Bestimmung von globaler Form und exakter Form
- Testen der Funktionalität: Erforschen der speziellen Funktion

- Prüfen der Beweglichkeit einzelner Teile.

Der Mensch benutzt diese Arten der Exploration sehr systematisch. Die gewonnenen Informationen gehen dabei über das allein durch Sehen Wahrnehmbare hinaus. Die Sinne ergänzen sich und sind keineswegs voneinander unabhängig. Die Performanz einer solchen Aktion ist von den Erfahrungen des Menschen abhängig, da Wissen über taktile Information implizit gespeichert ist und keine bewusste Verarbeitung erfordert. Eine unbekannte Form verlangt daher mehr Aufmerksamkeit als ein Objekt, welches sich in nur einem Merkmal von einem bekannten Gegenstand unterscheidet.

2.4.3.11 Sehen mit den Händen

Wie bereits erwähnt, ergänzen sich die Sinne, um ein möglichst vollständiges Bild eines wahrgenommenen Objekts zu gewinnen. In [89] geht man sogar einen Schritt weiter. Im dort aufgestellten Modell wird die Wahrnehmung der Hand als analog zum Sehen eines Objekts von der Rückseite definiert. Des Weiteren wird aus den gewonnenen Ergebnissen eine Blickpunkt-Abhängigkeit der haptischen Wahrnehmung abgeleitet, genau entgegen der visuellen Blickrichtung.

Möglicherweise versucht dieses Modell eine zu strikte Beschreibung der Zusammenhänge zwischen haptischer und visueller Wahrnehmung. Die Wichtigkeit der haptischen Wahrnehmung wird jedoch deutlich.

Unabhängig vom Zusammenspiel der einzelnen Sinne macht folgendes Beispiel die Mächtigkeit dieses Sinns deutlich: Ein blinder Mensch kann allein durch aktives Fühlen und Erkunden mit den Händen seine gesamte Umgebung wahrnehmen und im Detail beschreiben, abgesehen von der Farbe eines Objekts. Allerdings dauert dies deutlich länger als die visuelle Wahrnehmung, der Mensch ist und bleibt ein Augentier.

2.4.3.12 Psychomotorische Prozesse

Durch Erfahrung im Umgang mit dem eigenen Körper können Handlungen beschleunigt ausgeführt werden, da Aktionen bei entsprechender Darbietung von Reizen antizipiert werden können und den Körper auf die nächste Aktion vorbereiten. Ein Sportler erkennt durch eine Bewegung seines Gegners dessen Vorhaben, ein Autofahrer bewegt die Hand in Richtung des Schalthebels, wenn der Motor zu hoch dreht. Dies ist nur möglich durch die taktilen Rückmeldungen des Körpers, wodurch die eigene Körperhaltung ermittelt werden und der Körper in einer ähnlichen Situation wieder in eine ähnliche Haltung gebracht werden kann.

2.4.3.13 Interaktion der Hände mit Objekten

Das Ergreifen von Objekten mit den Händen ist ein hochkomplexer Vorgang, welcher von Seiten der Hardware möglichst genau simuliert werden muss. Dabei kommen alle Fähigkeiten der menschlichen Hand zum Einsatz. In [90] werden folgende Phasen der Interaktion unterschieden:

- Kontaktphase: Der Ort einer Berührung lässt sich aufgrund der hohen Rezeptordichte in den Fingern genau bestimmen. Leichte Berührungen der Haut führen nur zu einer Verformung, stärkerer Kontakt führt auch zu Veränderungen der Steifigkeit des Muskelapparates der Hand. Eine Untersuchung der Druckverteilung bei Berührungen findet man in [91].

- Ergreifen: Die Erscheinungsform eines Objekts sowie die beabsichtigte Art der Benutzung sind entscheidend für die Stellung der Finger während des Greifvorgangs. Man kann dabei zwei Arten des Greifens unterscheiden: Power-Grasping mit den Zielen Stabilität, Sicherheit und Umschließen mit der ganzen Hand; Precision-Grasping mit den Zielen Empfindlichkeit, Geschicklichkeit und Ergreifen mit den Fingerspitzen. Die Hand wird dabei vom Arm in die jeweils günstigste Position zur Durchführung der Aufgabe gebracht.
- Manipulieren: Beim Power-Grasping werden die Hände oft von den Armen unterstützt, das Objekt wird fest gegriffen. Beim Precision-Grasping hingegen kann das Objekt unter Einsatz der Finger bewegt oder abgetastet werden. Die aufgebrachten Kräfte liegen dabei nur geringfügig über der zum Halten des Objekts notwendigen Schwelle.
- Mechanischer Widerstand: Die Hand reagiert auf vom Objekt ausgeübte Kräfte beispielsweise durch Erhöhung der Muskelspannung. Dieser Reflex, welcher eigentlich dem Schutz vor Verletzungen (Überdehnung des Muskels) dient, kann bewusst zur Durchführung von Bewegungen eingesetzt werden.

Unterstützt wird das Benutzen der Hände wiederum vom visuellen System, welches vorab Informationen über die Lage des zu greifenden Objekts liefert, was wiederum eine Bereitstellung von räumlicher Information erforderlich macht.

2.4.4 Zusammenfassung

Im Kapitel zur *Interaktion* wurden die Grundlagen der Mensch-Maschine-Interaktion beschrieben, die für die Entwicklung eines neuen multimodalen Ein- /Ausgabegeräts notwendig sind, wie es später im Konzeptionskapitel beschrieben wird. Hierzu wurden drei Themen untersucht:

- 3-D-Interaktion
- Psychologie der virtuellen Realität
- Haptik.

Die Beschreibung der 3-D-Interaktion behandelte die Aspekte Skalierung, Level of Detail, Interaktionsmodell und Interaktionstechniken. Wichtigster Punkt dieses Unterkapitels war jedoch die Einführung des Begriffs „Metapher“, der später für das Konzept der „Virtual Glove Box“ wieder aufgenommen wird.

Im Unterkapitel zur Psychologie der virtuellen Realität wurden die Randbedingungen beschrieben, welche die Aktionen eines Benutzers in VR-Anwendungen steuern und beschränken. Die wichtigsten Konzepte dieses Kapitels betreffen das Design der Benutzungsschnittstelle, die Darstellung von Objekten in virtuellen Umgebungen und die Zusammenhänge zwischen Eingaben des Nutzers, Ausgaben des Systems und der Reaktion des Nutzers.

Das später vorgestellte Konzept zur Interaktion beschreibt ein neues Gerät, das neben visueller Ausgabe auch die Ein- und Ausgabe von Information über haptische Elemente vorsieht. Entsprechend beschreibt das Haptikkapitel die Besonderheiten, die aus Sicht der Mensch-Maschine-Interaktion an haptische Displays gestellt werden.

2.5. Computergestützte Biochemie

Begriffe wie „Bioinformatik“, „Bioinformatics“, „Computational Biology“ oder „computergestützte Biochemie“ sind gegenwärtig „Buzzwords“ und also solche in aller Munde. Wie in solchen Fällen üblich, gibt es bei aller Einigkeit über die Wichtigkeit des gemeinten Bereiches beträchtliche Unterschiede in der Meinung, was der Wesensgehalt oder Inhalt dieser Begriffe ist. Folglich existiert eine Reihe von (durchaus uneinheitlichen) Definitionsversuchen und Abgrenzungen.

Eine der weitesten Definitionen für „Computational Biology“ findet sich im „Webster's Online Dictionary: The Rosetta Edition“:

> *„a field of biology concerned with the development of techniques for the collection and manipulation of biological data, and the use of such data to make biological discoveries or predictions. This field encompasses all computational methods and theories applicable to molecular biology and areas of computer-based techniques for solving biological problems including manipulation of models and datasets.“* [92]

Die Beschreibung fasst sowohl den Bereich der Bioinformatik als auch den Bereich der computergestützten Biochemie. Die National Institutes of Health der USA [93] grenzt die beiden Bereiche wie folgt ab:

> *„Bioinformatics: Research, development, or application of computational tools and approaches for expanding the use of biological, medical, behavioral or health data, including those to acquire, store, organize, archive, analyze, or visualize such data.“*

Und:

> *„Computational Biology: The development and application of data-analytical and theoretical methods, mathematical modeling and computational simulation techniques to the study of biological, behavioral, and social systems“.* Beide Definitionen aus [94]

Obwohl auch hier die „Definitionen“ keine scharfen Abgrenzungen zulassen, wird doch ein Trend deutlich: Während sich die *Bioinformatik* vorzugsweise mit dem Speichern und Organisieren von Daten beschäftigt, ist die *computergestützte Biochemie* stärker an der Bildung von Modellen und an der Analyse biologischer Zusammenhänge interessiert. Als charakteristische Methode wird dabei die *Simulation* besonders hervorgehoben.

Die genannte Definition der computergestützten Biochemie bildet die Grundlage dieser Arbeit. Aber noch immer ist diese Definition recht weit gespannt, umfasst sie doch dynamische Systeme von molekularer Größenordnung (mit Strukturen im Größenbereich von Nanometern) bis zu sozialen Systemen, wie Schwärmen und Herdenn, aber auch Ausbreitungen von Krankheiten etc., die im Kilometermaßstab zu beschreiben sind. Aus dem weiten Gebiet biologischer Systeme sind bei einer Ordnung nach räumlicher Ausdehnung des Systems die drei „kleinsten“ für diese Arbeit interessant.

Die drei Bereiche sind

- das Molecular Modeling, also biochemische Systeme auf molekularer Ebene
- die Systembiologie, also die Teildisziplin der computergestützten Biologie, die als Zielsystem die Zelle hat
- die Zelldifferenzierung, also die Betrachtung des Verhaltens von Zellverbänden unterhalb der Bildung von Organen, zumeist in der embryonalen Forschung.

Diese drei Bereiche wurden aus zwei wesentlichen Gründen ausgewählt. Zum einen findet dort in besonderem Maße ein Umbruch von der „klassischen“ Laborwissenschaft hin zu einer computeruntestützten Wissenschaft statt, in der die Simulationstechnik eine immer wichtigere Rolle spielt. Zum anderen verlangt die räumliche Charakteristik der betrachteten Systeme nach entsprechenden Visualisierungs- und Interaktionstechniken, die mit der Simulationstechnik verbunden werden müssen. Im Folgenden sollen diese Bereiche kurz eingeführt werden.

2.5.1 Molecular Modeling

„Molecular Modeling“ beschreibt die Technik des Designs chemischer Verbindungen mit Hilfe des Computers. Bei dieser Molekülmodellierung will man Verbindungen mit gezielt gesteuerten Eigenschaften erhalten. Beispiele hierfür sind etwa das so genannte „Key-Lock-Problem“ [95] oder die Berechnung von Faltstrukturen von Proteinen. Beim „Key-Lock-Problem“ kennt man die Bindungsstelle eines (pharmakologisch interessanten) Moleküls (das „Schloss“) und sucht nun nach einem passenden Molekül, das an dieser Stell bindet (den „Schlüssel“).

Die traditionelle Notation für chemische Verbindungen als so genannte Strukturformel beschreibt ein Molekül als Kombination von Linien und Buchstaben, also als zweidimensionale Entität (siehe Abbildung 33).

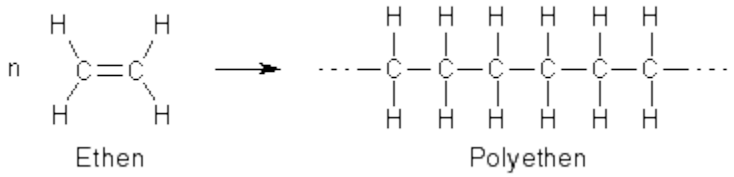


Abbildung 33 Strukturformel

Dabei wird jedoch vernachlässigt, dass die in der Biochemie und Pharmakologie betrachteten Moleküle nicht hinreichend als zweidimensionale Strukturen dargestellt werden können. Wichtige funktionale Eigenschaften dieser Verbindungen rühren vielmehr von ihrer räumlichen, also dreidimensionalen Struktur her. Das Hauptinteresse des „Molecular Modeling“ liegt daher in der dreidimensionalen Natur dieser relevanten Moleküle. Entsprechend muss auch die Visualisierung von Moleküldaten der 3-D-Struktur Rechnung tragen.

2.5.1.1 Molekül-Viewer in “Molecular-Modeling”-Anwendungen

Aufgrund der dreidimensionalen Natur des “Molecular-Modeling”-Problems benutzen existierende Anwendungen eine ähnliche Architektur [96], [97], [179]:

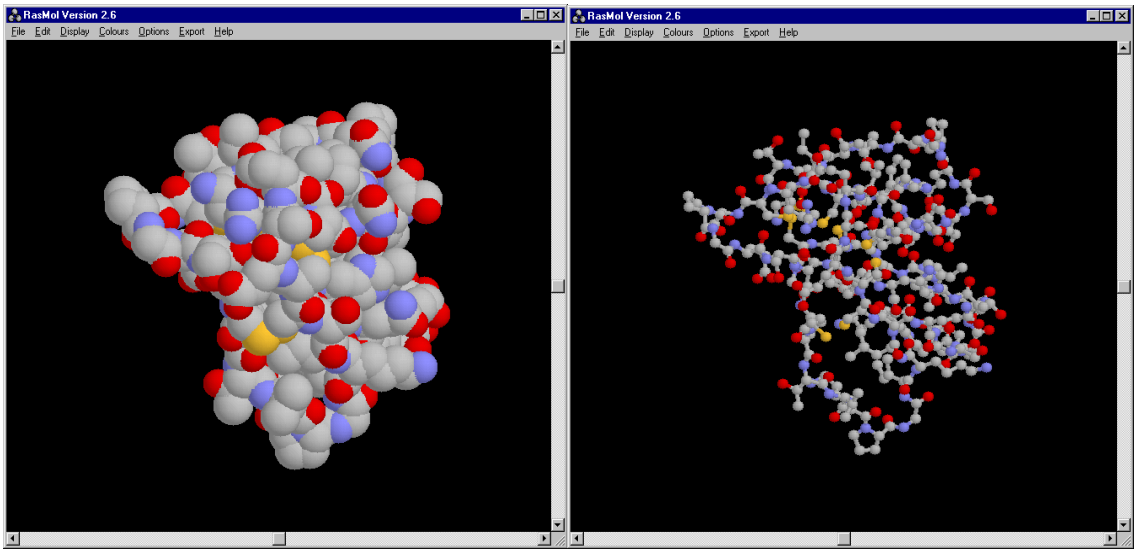


Abbildung 34 Verschiedene Visualsierungen mit dem RasMol-Visualisierungswerkzeug

Im Mittelpunkt der Anwendung steht ein Viewer, also eine Programmkomponente, die es erlaubt, die 3-D-Struktur eines Moleküls zu visualisieren (siehe Abbildung 34). Dieser Viewer ist ein reines Anzeigewerkzeug und in Bezug auf Simulationen oder Berechnungen hinsichtlich der Validität der dargestellten Strukturen nicht aktiv. Die Moleküldaten werden in der Regel über ein Datenfile eingelesen. Logisch um diese zentrale Einheit angeordnet sind dann funktionale Module für Berechnungen und Simulation. Der Datenaustausch zwischen den Modulen und dem Viewer verläuft in der Regel über einen Austausch von Datenfiles. Eines dieser Module ist oft ein graphischer Editor zur Erstellung neuer Moleküle. Auch dieses Modul hat in der Regel keine oder nur rudimentäre Validierungsmechanismen, so dass nicht garantiert wird, dass mit dem Editor erstellte „Moleküle“ überhaupt erzeugbar sind.

Derartige Validierungen und Berechnungen sind wiederum die Domäne anderer Module, die das chemische Verhalten der Moleküle simulieren. Die Simulationsergebnisse solcher Module werden zurück zum Viewer gegeben. Editiert der Benutzer diese Daten nun, werden die Simulationsergebnisse gegenstandslos, und die Simulation muss von neuem gestartet werden. Der „Molecular-Modeling“-Prozess kann entsprechend als Schleife beschrieben werden (siehe Abbildung 51): Der Benutzer editiert 3-D-Daten, die im Viewer visualisiert und von Simulationsmodulen validiert werden. Nach einem Simulationsdurchlauf nimmt der Benutzer weitere Veränderungen vor, und der Zyklus startet wieder.

Die Validierung oder die Simulation der Eigenschaften eines Moleküls kann von Bruchteilen von Sekunden bis zu einigen Stunden dauern. Während der Simulationsphase ist der Nutzer in der Regel zum Warten verurteilt.

Das Ergebnis einer Modellierungssession besteht normalerweise aus zwei Teilen:

1. Graphische Repräsentation der modellierten Stoffe (Standbild, Animation computergraphisches 3-D-File-Format, wie etwa VRML [98]. Diese Daten werden zur Präsentation und Dokumentation verwendet.)

2. Simulationsdaten, entweder in einem anwendungstypischen (oft proprietären) Format oder einem in der Biologie bzw. Chemie gebräuchlichen Format, das 3-D-Daten zulässt.

Zur Verdeutlichung sollen zwei gebräuchliche Dateiformate näher dargestellt werden.

2.5.1.2 Protein-Databank-File-Format (PDB)

Die Protein Data Bank [99] ist eine öffentlich zugängliche Datenbank für 3-D-Strukturdaten von Proteinen und Nukleinsäuren. Diese Daten werden der Datenbank von Wissenschaftlern aus der ganzen Welt zu Verfügung gestellt. Die Strukturaufklärung erfolgt in der Regel durch Röntgenkristallographie oder NMR-Spektroskopie [100]. Der Zugriff auf die Datenbank ist kostenlos.

Die Protein-Datenbank wurde 1971 als „Brookhaven Data Base“ am Brookhaven National Laboratory in den Vereinigten Staaten gegründet. 1998 ging die Datenbank in die Hände der „Research Collaboratory for Structural Bioinformatics (RCSB)“ über. Dies ist ein Gemeinschaftsprojekt der Rutgers University, der University of Wisconsin in Madison, des US National Institute for Standardisation (NIST) und des San Diego Supercomputer Center. Das Projekt wird von der US-Regierung (Department of Energy) und anderen US-Einrichtungen finanziert. Es besteht eine Zusammenarbeit mit dem Europäischen Bioinformatik-Institut und dem japanischen Institute for Protein Research.

Im Jahre 2002 enthielt die PDB etwa 18 000 Strukturen und erhält etwa 2 000 bis 3 000 weitere Einträge pro Jahr. Die Datensätze sind entweder im PDB-File-Format oder im so genannten mmCif-Format [101] gespeichert. Obwohl das mmCIF-Format das wesentlich jüngere und modernere Format ist, hat es lange nicht die Bedeutung seines Vorgängers erlangt, zumal es keine inhaltlichen Verbesserungen bietet. Im praktischen Einsatz dominiert das PDB-Format.

Beide Formate haben gemeinsam, dass sie neben der 3-D-Position der Atome auch Zusatzinformationen enthalten, wie etwa Autor bzw. einreichender Forscher, Veröffentlichungsort, aber auch sekundäre Strukturdaten, wie etwa die Aminosäurensequenz bei Proteinen.

In der PDB veröffentlichte Moleküle erhalten einen alphanumerischen Bezeichner, die so genannte PDB-ID. Diese ID bezeichnet die Einreichung eindeutig, nicht notwendigerweise jedoch das beschriebene Molekül. Ein Molekül in verschiedenen Konformationen oder ein in verschiedenen Umgebungen aufgeklärtes Molekül ist in der Regel unter verschiedenen PDB-ID zu finden. Jede Einreichung wird vor dem Eintrag von den Betreibern der Datenbank untersucht, annotiert und auf Plausibilität untersucht.

Das PDB-Datenformat besteht aus Datenzeilen, so genannten „Records“. Jede Zeile folgt einem strengen Schema, ist immer 80 Zeichen lang, und die ersten sechs Zeichen einer Zeile identifizieren den Record-Typ der Zeile. Obwohl ein Record über mehrere Zeilen reichen kann, sind die wichtigsten Records einzeilig. Im Anhang (Kapitel 9.2) ist ein Beispiel eines Records sowie eine Liste der gültigen Records zu finden.

2.5.1.3 MDL Chemical Table File Formats (CTfile Formats)

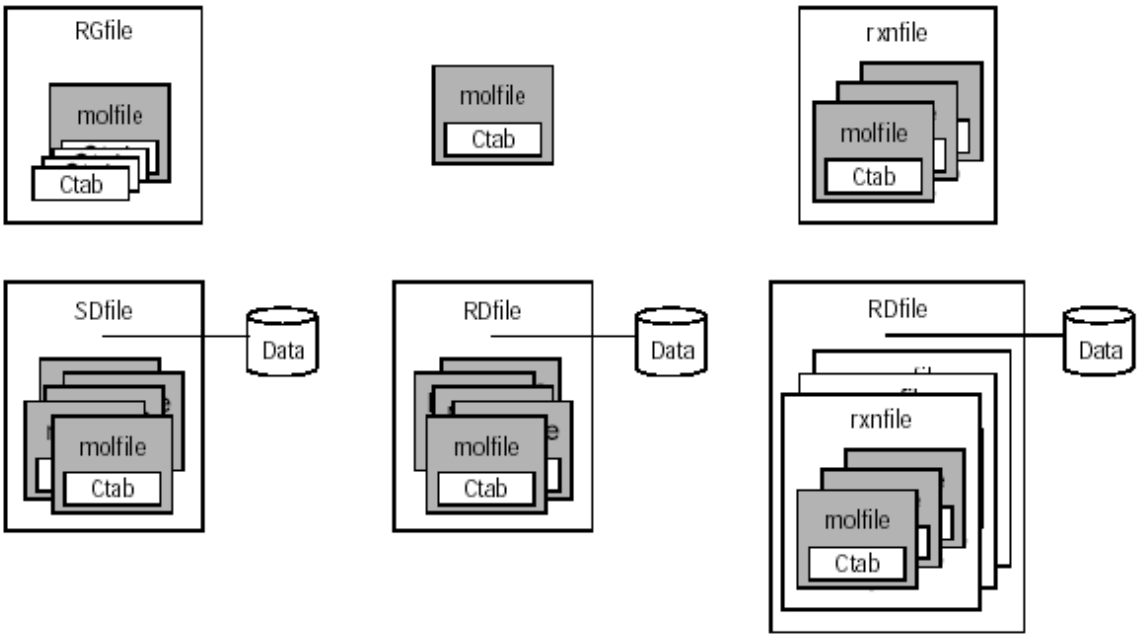


Abbildung 35 Fileformate von MDL [189]

Die Firma MDL hat eine Reihe von Fileformaten definiert, die den Datenaustausch und die Datenpersistenz der von dieser Firma vertriebenen Werkzeuge erleichtern sollen. Die Offenlegung der Formate sorgte für eine weitgehende Verbreitung.

Zentrales Format der Fileformatfamilie ist das „Molfile"-Format. Abbildung 36 zeigt ein Beispiel. Die folgenden Fileformate sind hierauf aufgebaut (Abbildung 35):

- *Molfile* enthält ein einzelnes Molekül.
- *RGfile* enthält neben dem Ausgangsmolekül „Rgroups" als Ergebnis einer Abfrage. (Dieses Format hat nur im MDL-Progammkontext Bedeutung.)
- *Rxnfiles*: Jedes Rxnfile enthält Strukturinformationen zu den Edukten und Produkten einer einzelnen Reaktion.
- *SDfile*: SDfiles enthalten eine beliebige Anzahl von Molekülstrukturdaten (jeweils in Molfiles beschrieben). Darüber hinaus lässt sich das Formatfile durch beliebige weitere Daten anreichern.
- *RDfile*: RDfiles erweitern die Möglichkeiten der SDfiles durch die Möglichkeit, auch Reaktionsinformationen (Rxnfiles) zu speichern.

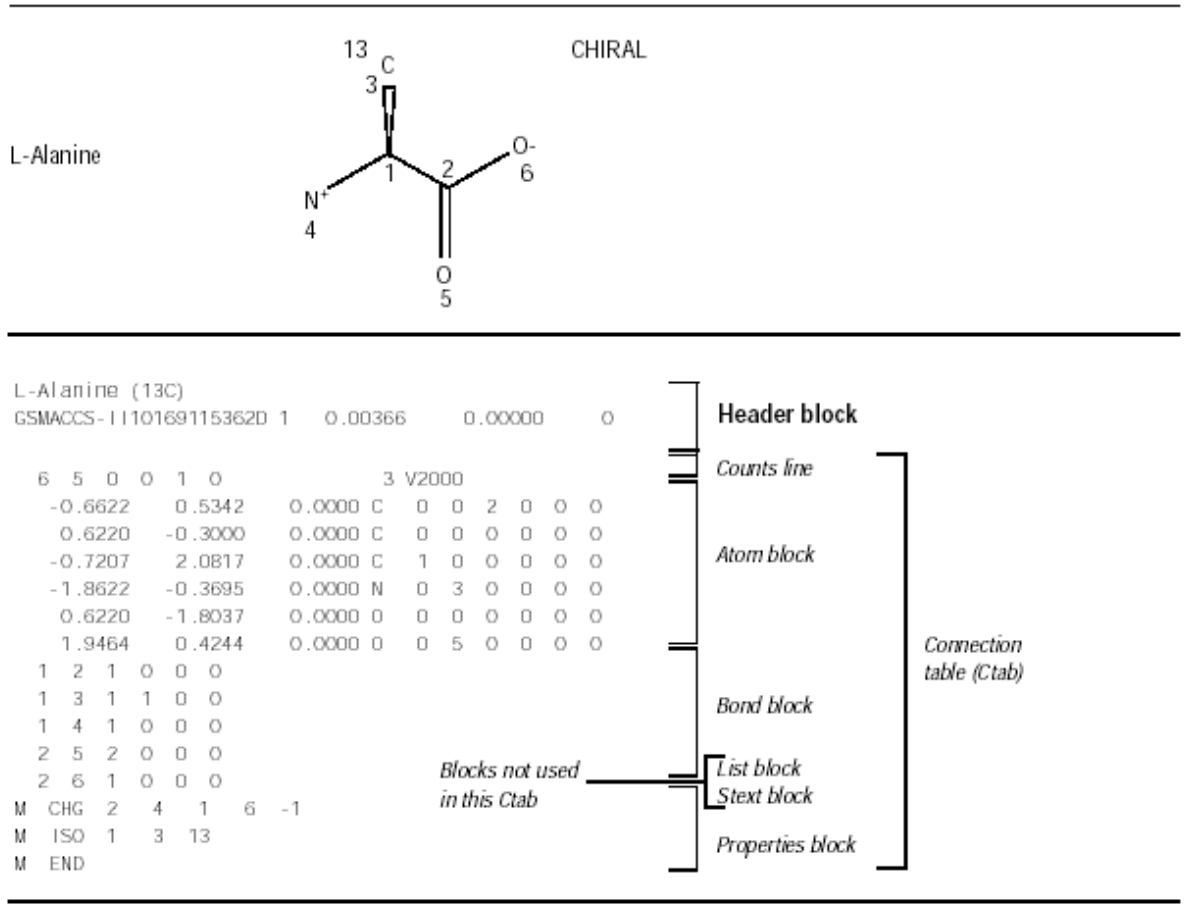


Abbildung 36 Beispiel eines Molfiles

2.5.2 Systembiologie

Die Zellbiologie hat als Teildisziplin der Biologie in den letzten 20 Jahren einen großen Wandel erfahren. Während sich das grundlegende Ziel, die Aufklärung der chemischen und physikalischen Prozesse, die das Leben ausmachen, nicht geändert hat, haben sich die Methoden stark verändert. Mit Hilfe konfokaler Mikroskopie, Zwei-Photon-Fluoreszenz-Mikroskopie, Ultraschallmikroskopie [102] und ähnlicher Methoden lassen sich heute die interessierenden Prozesse *in situ* beobachten, also in und an der lebenden Zelle selbst. Neben den mikroskopischen Techniken haben sich entsprechend auch andere sensorischen Techniken verbessert. Das Besondere dieser neuen Situation ist, dass es nun möglich ist, die Zelle als *System* statt als Sammlung einzelner Phänomene und Teilprozessen zu sehen.

Gleichzeitig mit dieser Entwicklung hat sich das Wissen über die in den dynamischen Prozessen beteiligten Strukturen und Molekülen entscheidend erweitert. Dieses Wissen reicht von großen Datenbanken von Proteinstrukturen [99] über die Aufklärung gesamter Genome verschiedener Organismen bis hin zur Aufklärung des menschlichen Genoms [103].

All diese neuen Forschungsbereiche (oft als *Omics,* von Gen*omics,* Prote*omics* etc.bezeichnet) haben eine Flut von Daten erzeugt, deren Bewältigung für die Biologie zunehmend zum Problem wurde. Aus dieser Anforderung entwickelte sich auf Seiten der Informatik die

Teildisziplin der *Bioinformatik*, um die computergestützten Werkzeuge zur Handhabung zu entwickeln.

In dieser Situation fällt es zunehmend schwerer, den Überblick über die komplexen Dynamiken zu behalten und neue Erkenntnisse zu gewinnen. Abhilfe sollen hier der Einsatz der Simulation und der damit zusammenhängende Einsatz ganzheitlicher Simulationsmodelle bringen. Obwohl Simulationsstudien in vielen Ingenieurdisziplinen schon lange Zeit zum Werkzeugarsenal der Wissenschaftler gehören, waren sie in der Biologie im zellularen Maßstab wenig bekannt und gewinnen in den letzten Jahren unter der Überschrift *Systembiologie* und *Computational Biology* zunehmend an Bedeutung [104].

Das Hauptziel der *Computational Cell Biology* ist es, dem Biologen Werkzeuge an die Hand zu geben, die es ihm erlauben, Modelle über die inter- und intrazellulären Prozesse zu machen. Die erstellten Modelle beschäftigen sich mit Hypothesen über die in der Zelle vorhandenen chemischen Substanzen und deren biochemischen oder physikalischen Transformationen.

Der Erkenntnisgewinn mit Hilfe von biologischen Simulationsstudien geschieht im Abgleich der durch die Simulation gewonnenen Daten mit realen, im Labor gefundenen Daten. Es ist gerade dieser Prozess des Transfers von Wissen aus Experimenten in das Modell und insbesondere aus der Simulation in das Experiment, der Gegenstand der gegenwärtigen wissenschaftlichen Diskussion ist (siehe Abbildung 37) [104].

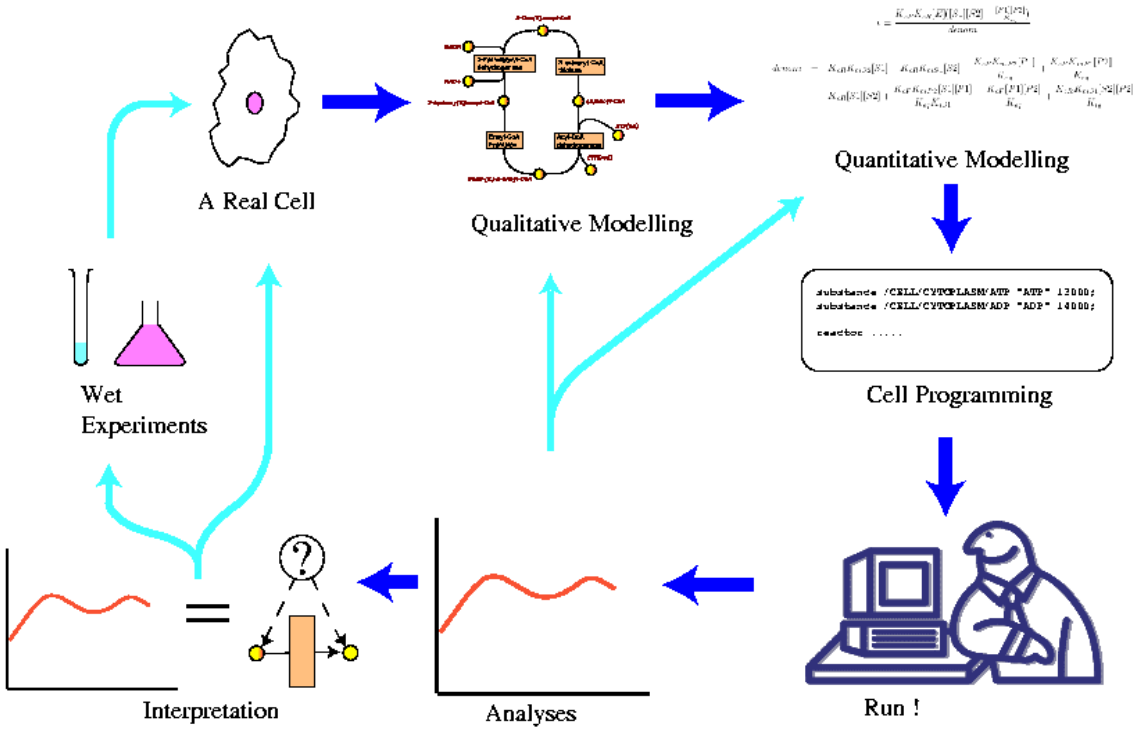


Abbildung 37 E-Cell Cycle [104]

In der Vergangenheit wurden Simulationsstudien (inklusive der Erstellung der Simulationssoftware, des Simulationsmodells und der Ergebnisvisualisierung) sehr oft für sehr begrenzte Einzelfälle erstellt und durchgeführt [105]. Hierdurch wurden zwar dedizierte Simulatoren entwickelt, die jedoch, wie für Insellösungen typisch, nicht in einen größeren Zusammenhang gestellt werden können [106].

Andere Wege geht eine Reihe von großen Projekten, die auf das Spezialwissen der Einzelstudien zugunsten eines generischen und oft auch ganzheitlichen Ansatzes verzichten. Im Folgenden sollen einige der wichtigsten dieser Projekte näher beschrieben werden.

2.5.2.1 Virtual Cell

Virtual Cell ist ein am „University of Connecticut Health Center“ gestartetes Projekt mit dem Ziel, eine Softwareumgebung zu schaffen, die als Unterstützung der laborgestützten Zellforschung dienen soll [106]. Auf das Werkzeug kann über das Internet mittels eines normalen Webbrowsers zugegriffen werden [107]. Eine wichtige Prämisse des Projektes ist, dass die mathematisch-physikalische Ausbildung der Forscher in der Laborbiologie nicht ausreichend ist, die komplexen theoretischen Systemmodelle selbst zu erstellen. Entsprechend unterscheidet Virtual Cell zwei Benutzer- bzw. Autorenrollen im Umgang mit systembiologischen Simulationswerkzeugen:

- den Laborwissenschaftler
- den Modellierer.

Für beide Rollen werden jeweils getrennte Arbeitsbereiche zu Verfügung gestellt. Der *BioModel*-Arbeitsbereich soll es dem Laborwissenschaftler ermöglichen, Modellkomponenten durch die Eingabe von Kompartimenten, Molekül-Spezies und Reaktionskinetiken zu definieren. Der *Mathworkspace* für den mathematisch orientierten Modellierer erlaubt die Modellbildung mit Hilfe einer mathematischen Beschreibungssprache [108]. Modelle aus dem *BioModel* können automatisch in den *Mathworkspace* konvertiert werden und umgedreht. So soll ein Datenaustausch zwischen den Benutzern ermöglicht werden.

2.5.2.2 E-Cell

E-Cell ist ein 1996 am japanischen Institute for Advanced Biosciences, Keio University initiiertes Projekt mit dem Ziel, eine Simulationsumgebung zur Erforschung von intrazellularen Dynamiken zu schaffen [109]. Tatsächlich ist das System nicht auf bestimmte Vorgänge limitiert. Vielmehr sollte durch ein konsequent objektorientiertes Konzept [8] ein Framework erstellt werden, mit dem es möglich ist, verschiedene algorithmische Ansätze zu erproben.

Der wichtigste Entwicklungsschritt der derzeit entwickelten Version 3 des E-Cell-Systems ist die gleichzeitige Unterstützung von mehreren Algorithmen in gegebenenfalls verschiedenen Zeitskalen. Dabei wird der Ansatz von verschiedenen diskreten Zeitgebern gewählt, so genannten *Steppern* Diese Zeitgeber werden über einen Scheduler verwaltet, der die verschiedenen wartenden Stepper aktiviert. Die Stepper erzeugen Ereignisse, welche wiederum von den zugeordneten Zeitgebern aufgenommen werden und einen Verarbeitungsschritt einleiten. Um in einem solchen diskreten ereignisbasierten System kontinuierliche Prozesse abbilden zu können, müssen diese allerdings über Integratoren in ein diskretes ereignisbasiertes System überführt werden (Abbildung 38).

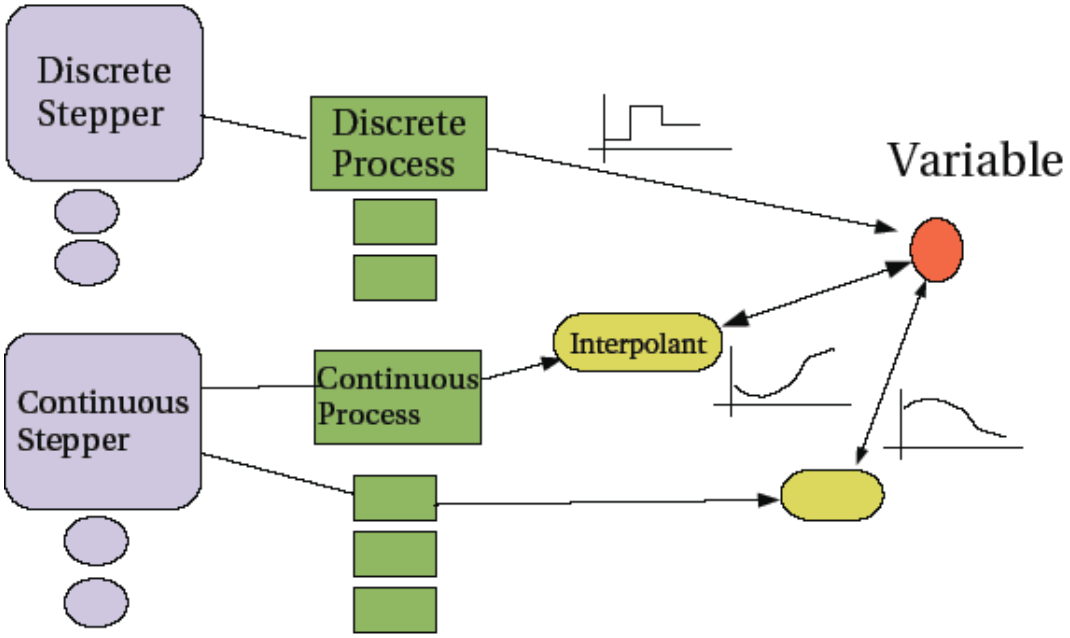


Abbildung 38 E-CELL-Stepper [104]

Das E-Cell-System (in seiner ersten Inkarnation) wurde in einer Reihe von Anwendungsstudien validiert, so etwa in einer Untersuchung zu Diabetes [110] oder Untersuchungen zu menschlichen Erythrozyten und myocardialen Zellen [111].

2.5.2.3 Project Cybercell

Project Cybercell [112] ist ein von der University of Alberta in Kanada ausgehendes Verbundprojekt verschiedener kanadischer Universitäten. Wie die anderen genannten Initiativen will Project Cybercell eine virtuelle Experimentalumgebung schaffen, die es erlaubt, genaue Untersuchungen zum Verhalten lebender Zellen durchzuführen. Diese Umgebung stützt sich ebenfalls auf die Simulation als zugrunde liegendes Konzept.

Anders als die anderen, eher softwaretechnisch motivierten Projekte will Project Cybercell aus experimentellen Daten „Bottom-up"-Modelle generieren und diese in Animationen („Movies") visualisieren (siehe Abbildung 39).

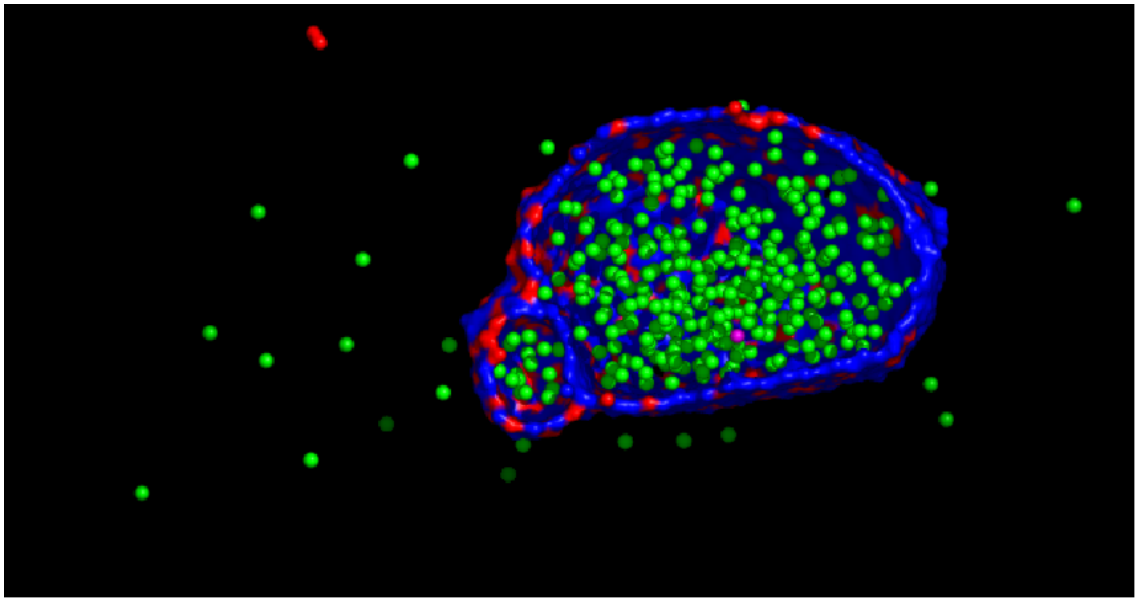


Abbildung 39 ProjectCybercell [112]

Das Projekt hat E. coli als Modellorganismus ausgewählt und will damit Konzepte und Methoden erforschen, die später auf kompliziertere Zelltypen erweitert werden sollen.

2.5.2.4 Projektübergreifende Initiativen

Eine wichtige Erkenntnis aus den bisher exemplarisch vorgestellten Initiativen ist, dass sich die Ansätze der verschiedenen Projekte in der Regel wenig durch die verwendeten Simulationsalgorithmen unterscheiden; es finden immer entweder auf Gillespie zurückzuführende stochastische ereignisbasierte Methoden [113] oder auf gewöhnliche Differenzialgleichungen basierende Methoden Anwendung. Unterschiede finden sich lediglich in der Auswahl dieser Algorithmen, des softwaretechnischen Designs und der Handhabbarkeit der Werkzeuge. Einer der wichtigsten Aspekte der Systembiologie ist dabei jedoch die Erstellung und Validierung neuer Systemmodelle. Ein aussagefähiger Vergleich der verschiedenen Softwarewerkzeuge wird also erst durch die Austauschbarkeit von Modellen möglich werden.

Aus dieser Erkenntnis heraus bildete sich eine Initiative mit dem Ziel, eine größere Interoperabilität zwischen verschieden Werkzeugen zu erreichen. Ergebnis dieser Initiative sind zwei Projekte:

- *Systems-Biology-Workbench* (SBW) als Softwareframework
- *Systems-Biology-Markup-Language* (SBML) als Modellbeschreibungssprache.

Im Folgenden soll zunächst die Systems Biology Workbench betrachtet werden.

2.5.2.4.1 Systems-Biology-Workbench

Das Ziel der *ERATO-Systems-Biology-Workbench* (SBW) ist es, ein offenes Softwareframework für die Einbindung verschiedener unabhängiger Werkzeuge im Kontext der Systembiologie zu entwickeln [114]. Zu diesem Zweck wurde eine Open-Source-Bibliothek entwickelt, die diese Einbindung über eine gemeinsame Programmierschnittstelle ermöglicht.

Die SBW-Architektur wurde bisher auf so unterschiedliche Plattformen wie Linux, FreeBSD und Windows (98, 2000, XP) portiert und unterstützt verschiedene Programmiersprachen, darunter C, C++, Java und Python.

Das Design der SBW soll es dem Nutzer ermöglichen, verschiedene „SBW-enabled" Softwarewerkzeuge miteinander zu verbinden. Diese Funktionen sollen für den Benutzer so transparent wie möglich gemacht werden.

Vom Standpunkt des Softwareentwicklers aus basiert die SBW auf einer Brokerarchitektur (siehe Abbildung 40).

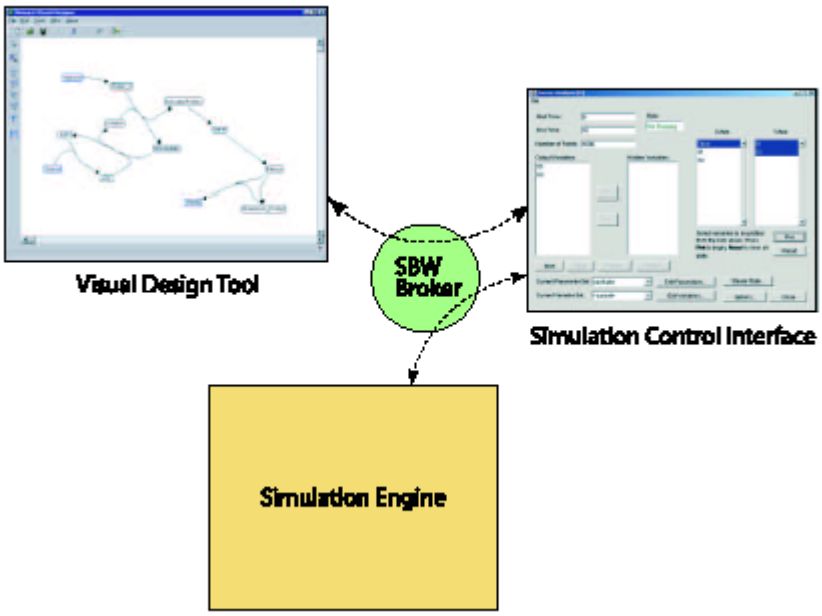


Abbildung 40 SBW Broker [114]

Über diesen Broker finden sich und kommunizieren die verschiedenen Module, sprich Programme, miteinander. Als Kommunikationsmedium werden dabei leichtgewichtige Strukturen wie etwa Network Sockets verwendet. Die Kommunikation mit dem Broker, etwa um andere Module zu finden und deren Fähigkeiten abzufragen, aber auch zwischen den Modulen, basiert auf einem Messagekonzept. Dies bedeutet, die verschiedenen Applikationen tauschen wohldefinierte Datenstrukturen, so genannte Messages aus. Um die Arbeit mit der SBW zu erleichtern, bietet das Framework dem Entwickler zwei verschiedene Programmierschnittstellen an: eine Low-Level-API, die es erlaubt, auf Verbindungsebene auf SBW-Funktionalitäten zuzugreifen, und eine High-Level-API, die einige Komplexitäten vor dem Entwickler auf Kosten der Flexibilität verbirgt. Die Funktionalität des Frameworks basiert auf dem etwa aus JavaRMI [114], CORBA [115] bekannten Prinzip des Aufrufens entfernter Funktionen [116]. Das folgende Programmfragment verdeutlicht dies.

```
SBWDouble doTrigonometry(SBWDouble x)
{
      try
```

```
    {
        // Start a new instance of the trigonometry module.
        Module module =
        SBW::getModuleInstance("edu.caltech.trigonometry");
        // Locate the service that we want to call in the
        // module.
        Service service = module.findServiceByName("trig");
        Method method = service.getMethod("sin");
        // Make the call and get the result back.
        DataBlockReader resultData =
        method.call(DataBlockWriter() << x);
        // Extract the result data.
        SBWDouble result;
        resultData >> result;
        return result;
    }catch (SBWException e)
    { e.HandleWithDialog();
    }
    return 0;
}
```

Tabelle 3 SBW Fragment

2.5.2.4.2 Systems-Biology-Markup-Language (SBML)

Während sich die *Systems-Biology-Workbench* mit der Interoperabilität von Softwarewerkzeugen auf Anwendungsniveau beschäftigt, richtet sich die *Systems-Biology-Markup-Language (SBML)* [117] an die Austauschbarkeit der Simulationsmodelle. Die SBML soll dabei geeignet sein, Modelle biochemischer Reaktionsnetzwerke zu beschreiben, wie sie in typischen systembiologischen Fragestellungen auftreten, z. B. bei der Aufklärung metabolischer Pfade oder Signaltransduktionspfaden. Die SBML ist nach dem Willen ihrer Entwickler systemunabhängig und nutzt daher XML, die eXtensible Markup-Language [118], als Basis ihrer Beschreibung [119]. Die SBML liegt gegenwärtig als Level 2 vor[113]. Obwohl sich Level-1-Dokumente problemlos in Level-2-Dokumente konvertieren lassen, ist Level 2 nicht abwärtskompatibel.

In das Design der SBML sind Erfahrungen und Anforderungen einer Reihe von Werkzeugen eingegangen:

- BASIS [120],
- Bio Skektch Pad [121],
- BioSpreadsheet [122],
- Gepasi [126],
- Jarnac [129],
- JDesigner [130],

- BioSpice [123],
- CellDesigner [124],
- Cellerator [125],
- COPASI [126],
- DBsolve [127],
- E-CELL [109],
- ESS [128],
- JigCell [131],
- MCell [132],
- Net-Builder [133],
- PathScout [134],
- ProMoT/DIVA [135],
- StochSim [119],
- Virtual Cell [106]

Darüber hinaus gab es Mithilfe von Autoren der CellML [136]. Der Standardisierungsprozess wird kontinuierlich fortgeführt, und es wird erwartet, dass auch zukünftige Entwicklungen auf das Aussehen der SBML Einfluss nehmen.

Im Folgenden soll eine kurze Beschreibung der Grundideen der SBML gegeben werden. In Abbildung 41 sei ein einfaches Beispiel eines biochemischen Reaktionsnetzwerkes gegeben.

$$X_0 \overset{k_1}{\Rightarrow} S_1$$
$$S_1 \overset{k_2}{\Rightarrow} X_1$$
$$S_1 \overset{k_3}{\Rightarrow} X_2$$

Abbildung 41 Reaktion SBML

Ein solches Reaktionsnetzwerk enthält verschiedene Komponenten, für die die SBML die jeweiligen Beschreibungskonstrukte zu Verfügung stellt. Die Hauptbestandteile sind:

- Ausgangsstoffe (reactant species) der Reaktion
- Reaktionsprodukte (product species)
- Reaktionen
- Reaktionskinetiken (rate laws) mit ihren Parametern.

Darüber hinaus sind Beschreibungen der Zellkompartimente (etwa Cytoplasma, Zellkern etc.) und der verwendeten Einheiten der beschriebenen Größen notwendig. Die Grundstruktur eines SBML-Modells ist entsprechend:

```
Anfang der Definition
     Liste von Funktionen
     Liste von Einheiten(-definitionen)
     Liste von Kompartimenten
     Liste von Stoffen
     Liste von Parametern
     Liste von Regeln
     Liste von Reaktionen
     Liste von Ereignissen
Ende der Definition
```

Tabelle 4 SBML Modellstruktur

Jede dieser Listen ist dabei optional. Während es allerdings für Listen wie die der Einheiten implizite Standardlisten gibt, ist ein Modell ohne Stoffe oder Reaktionen im Allgemeinen wenig sinnvoll. Für die Beschreibung von Reaktionskinetiken und allgemein für die Beschreibung mathematischer Ausdrücke verwendet die SBML die MathML [137], eine Mark-up-Language für die Beschreibung solcher Ausdrücke. Die MathML ist unabhängig vom Anwendungsfeld, hier der Systembiologie. Ein Beispiel, welches das oben dargestellte Reaktionsnetzwerk (Abbildung 41) beschreibt, ist im Anhang (Kapitel 9.3) gegeben.

2.5.2.4.3 CellML

Die Motivation und der Lösungsansatz der CellML [136] gleicht der der SBML. Auch hier soll eine auf XML basierende Markup-Sprache biochemische Reaktionsmodelle beschreiben und so den Austausch von Modellen ermöglichen. Anders als die SBML wird CellML maßgeblich von einem kommerziell agierenden Unternehmen beeinflusst, der Physiome Sciences Inc. [138].

Eine konzeptionelle Besonderheit der CellML ist die Definition von Variablen mit lokalem Gültigkeitsbereich. Variablen gelten nur innerhalb einer per `<component>`-Tag definierten Komponente, etwa eines Kompartimentes. Derartige Variablen können mit einer *Interface*-Deklaration nach außen verfügbar gemacht werden. „Außen“ meint hier entweder weitere Komponenten innerhalb der gleichen Komponentengruppe oder die nächsthöhere Hierarchiestufe. Um eine Verbindung zwischen zwei Komponenten herzustellen, lassen sich Variablen über ihre *Interfaces* verbinden (`<connection>`). Die Intention dieser Konstruktion ist es, Modelle verschiedener Modellierungsgranularitäten zusammenzuschließen. Im Anhang ist ein Beispiel für die Verwendung der CellML gegeben (Kapitel 9.4).

Obwohl Herangehensweise und Konzept der CelllML und der SBML sich ähneln, hat die SBML eine wesentlich größere Verbreitung erfahren. Dies ist nach Aussagen der Initiatoren der SBML vor allem auf das von Anfang an größere Angebot an SBML- und SBW-kompatibler Software zurückzuführen.

2.5.3 Zelldifferenzierung

Obwohl der Begriff der Systembiologie in den meisten Fällen für intrazelluläre Vorgänge verwendet wird, lässt er sich auch auf interzelluläre Vorgänge und damit auf das Verhalten von Zellverbänden erweitern. Von besonderer Bedeutung sind dabei die „Zellverbände“ höherer Organismen, die wiederum eigene Systeme, die Organe, bilden. Die Entwicklung multizellularer Organismen ist ein Prozess, der in der Biologie von besonderem Forschungsinteresse ist. Dies gilt insbesondere, da jede Zelle den gleichen kompletten Satz von Genomen enthält und sich dennoch die unterschiedlichsten Zellen aus einer gemeinsamen Stammzelle differenzieren [184].

Haben sich erst einmal differenzierte Zellen gebildet, formieren gleichartige Zellen oft einen Verband. Dabei ist die Funktion des Verbands mehr als nur die parallele Ausführung gleichartiger „Programme“ [139], es entstehen vielmehr Kommunikationsbeziehungen und gegenseitige Abhängigkeiten [140].

Der derzeitige Stand der Forschung geht von drei Faktoren aus, die die Differenzierung und Zusammenarbeit zwischen den Zellen beeinflussen:

- Wichtigster Faktor ist die intrazellulare Dynamik der Zelle, wie sie bereits im Kapitel zur Systembiologie beschrieben wurde. Zu beachten ist dabei, dass der Detailgrad der Modellierung dieser Dynamik in der praktischen Umsetzung geringer ist als die Modelle zur Erforschung einer einzelnen Zelle [140].

- Interzellulare Interaktion zwischen Zellen durch den Austausch chemischer Stoffe. Alle lebenden Zellen geben Stoffwechselprodukte an das interzellulare Medium ab. Manche dieser Stoffe wirken als Botenstoffe und beeinflussen den inneren Zustand anderer Zellen. Man unterscheidet hier Stoffe, die die Zellmembran durchdringen und so direkt selbst den Systemzustand der aufnehmenden Zelle verändern, und Stoffe, die an der Zelloberfläche von speziellen Molekülen (so genannten Rezeptoren) gebunden werden. Die Bindung über Rezeptoren löst innerhalb der Zelle eine Reaktionskaskade innerhalb eines Reaktionsnetzwerks aus. Dieser Vorgang der so genannten „Signaltransduktion" ist ein eigenständiges Gebiet innerhalb der Systembiologie.
- Zelladhesion, also das Anhaften von Zellen untereinander und die damit verbundene lokale Fixierung der Zellen, zusammen mit den dabei ausgeübten mechanischen Kräften, ist ein noch wenig betrachteter Faktor [141].

2.5.4 Visualisierung in der computergestützten Biochemie

Alle drei der oben beschriebenen Bereiche zeichnen sich durch eine große Menge stark interdependenter Daten aus. Ein Mittel, dieser „Datenflut" Herr zu werden, ist der Einsatz von Visualisierung. Naturgemäß fällt die Wahl der Mittel für die verschiedenen Bereiche unterschiedlich aus.

Im Bereich des Molecular Modeling ist eine 3-D-Darstellung von Atomen, Molekülen und z. B. Oberflächen Stand der Technik (siehe Abbildung 42 und Abbildung 43); die direkte interaktive Einbindung von Simulation ist jedoch nicht sehr weit gediehen.

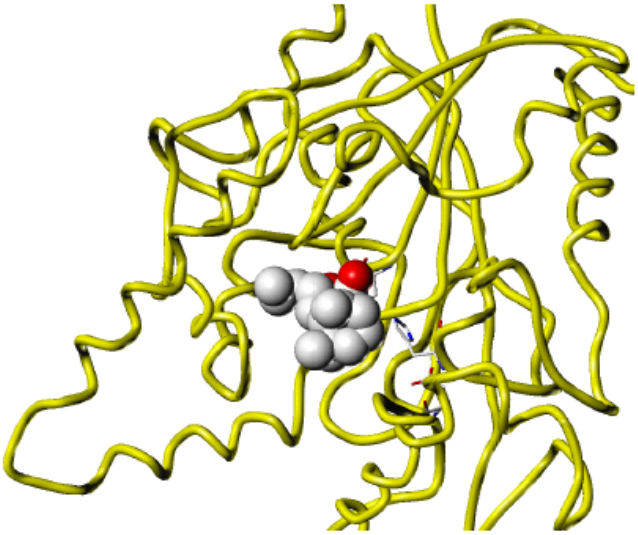


Abbildung 42 Cerius2 [96]

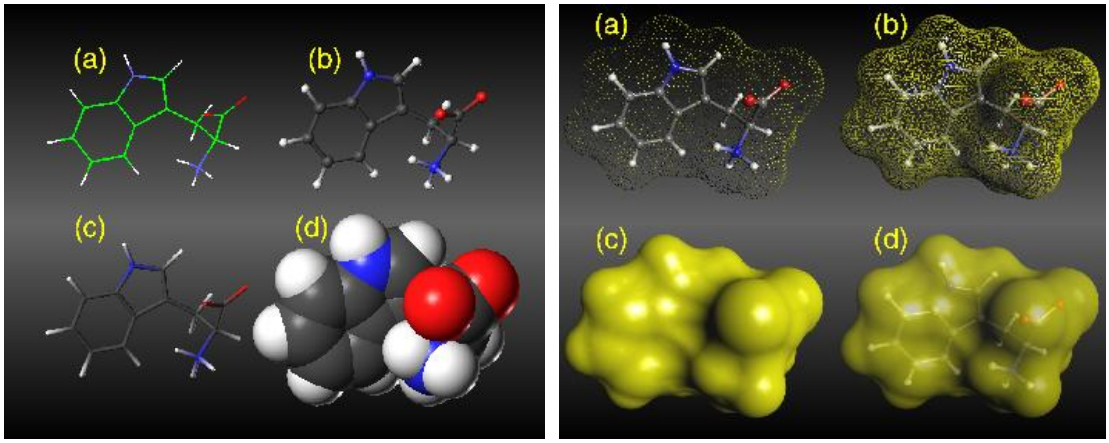


Abbildung 43 MolCAD [97]

In der Systembiologie zeigt sich auch in der Visualisierung die Fokussierung der Forschung auf die Aufklärung von Netzwerken unter Missachtung der dreidimensionalen Natur des Modellsubjekts „Zelle“: Abbildung 44, Abbildung 45 und Abbildung 46.

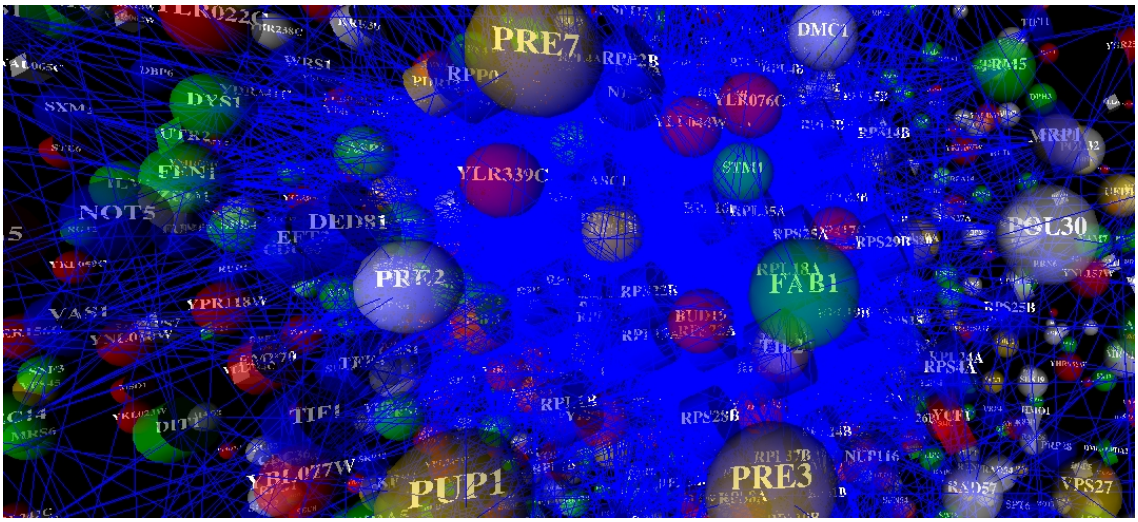


Abbildung 44 Protein – Protein Interaktionsnetzwerk [185]

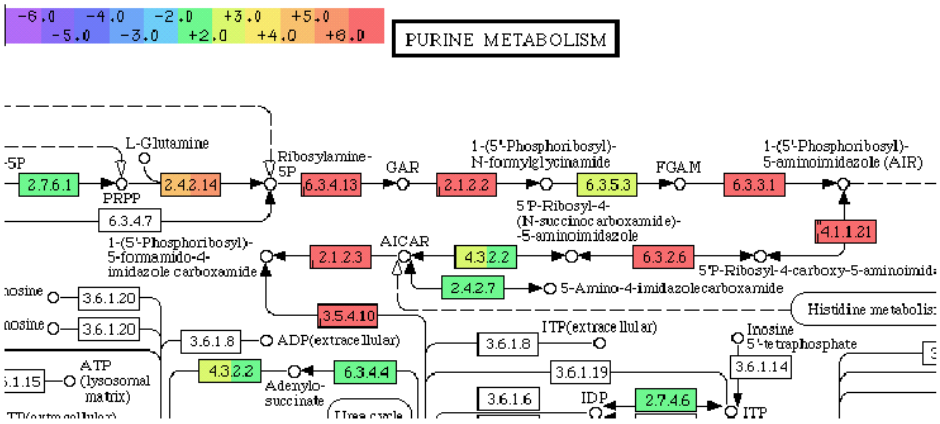


Abbildung 45 Ausschnitt eines metabolischen Netzwerks

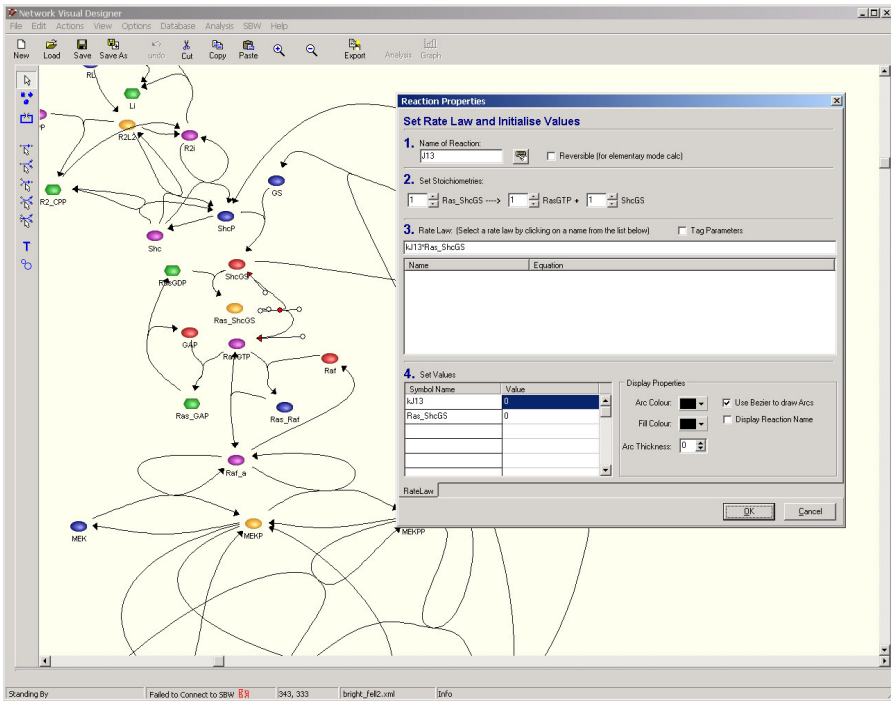


Abbildung 46 Regulationsnetzwerk [130]

Die Visualisierung im Bereich Zellverbände und Zelldifferenzierung kann den räumlichen Charakter des betrachteten Subjekts nicht ignorieren. Tatsächlich finden sich wenige Visualisierungen dieses Bereichs, die dann auch eher Abbildungen in 2D beinhalten. Selten werden 3-D-Visualisierungen verwendet (siehe Abbildung 47 und Abbildung 48).

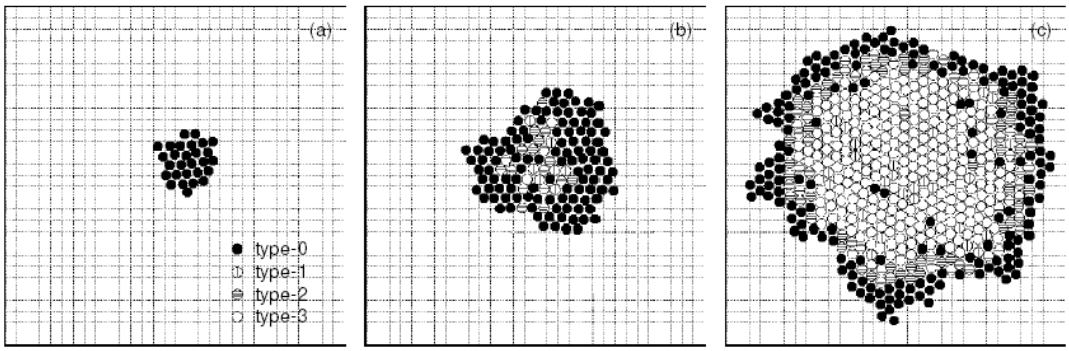


Abbildung 47 2D Zelldifferenzierungssimulation [135]

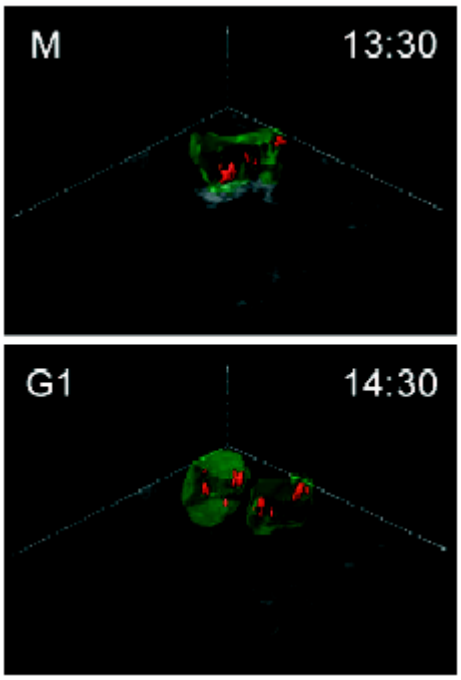


Abbildung 48 3-D-Zelldifferenzierung [143]

2.5.5 Zusammenfassung

In der Zusammenschau dieses Kapitels zeigt sich, wie ähnlich bei aller Unterschiedlichkeit in der Materie die Probleme im Bereich der computergestützten Biochemie sind. Für alle gezeigten Bereiche gilt, dass die Biologie bzw. die Chemie das Anwendungsgebiet stellt und mit ihren klassischen Methoden an Grenzen stößt. Die beigezogene Hilfswissenschaft ist in der momentanen Phase die Simulationstechnik. Hier gilt, dass sich die computergestützte Biochemie gegenwärtig in einer Lage befindet, in der die Erkenntnisse der unterstützenden Wissenschaften für das Anwendungsgebiet nachvollzogen und adaptiert werden. Im Bereich der Simulation sind das die wohlbekannten Techniken der Lösung von (gewöhnlichen oder partiellen) Differenzialgleichungssystemen oder die Nutzung von stochastischen Methoden, z.

B. im Zusammenhang mit Monte-Carlo-Methoden. Diese Techniken sind in Kapitel 2.2.2.5 näher beschrieben.

Obwohl es gegenwärtig den Anschein hat, dass diese bekannten Methoden ausreichend sind, um die gestellten Probleme des Anwendungsgebiets prinzipiell zu bearbeiten („prinzipiell“, da die inhärente Komplexität der meisten relevanten Probleme eher nach Lösungen der Rechnertechnik, wie etwa Parallelisierungen und verteiltem Rechnen, verlangen denn nach mathematisch neuen Verfahren), zeigt die praktische Diskussion [142] mit den aktuell Handelnden, dass die dringlichsten praktischen Probleme auf einem anderen Gebiet liegen:

Es fehlt an der Integration von interaktiven Technologien, die es erlauben, mit den Simulationsergebnissen, den Simulationsmodellen oder den Messdaten auf angemessene Weise zu interagieren. Dies drückt sich unter anderem an einem Mangel adäquater Visualisierungen der Ergebnisse aus. Im Folgenden sollen hieraus Anforderungen für die interaktive Simulation und Visualisierung in der computergestützten Biochemie abgeleitet werden.

3. Anforderungsanalyse

Die Anforderungen an eine interaktive Visualisierung und Simulation in der computergestützten Biochemie kommen natürlich vor allem aus dem Anwendungsfall. Dennoch ergeben sich aus dem Zusammenspiel von Simulation und Visualisierung Anforderungen, die über das Anwendungsfeld hinausgehen. Im Folgenden soll zunächst der Zusammenhang von Simulation und Visualisierung in diesem Bereich herausgestellt werden. Die zusätzlichen Anforderungen ergeben sich durch die Forderung der Interaktivität. Im Anschluss sollen die spezifischen Anforderungen an *Simulation* und an *Interaktivität und Visualisierung* genauer betrachtet werden.

Im Kapitel zur Simulationstechnik (Kapitel 2.2) wurde beschrieben, dass die Durchführung einer Simulationsstudie der Durchführung einer Experimentalstudie ähnelt. Die Simulationsstudie zeichnet sich lediglich durch eine „Versuchsdurchführung“ allein im Rechner aus. Hieraus ergeben sich sowohl Gemeinsamkeiten als auch Unterschiede im Verhältnis von Simulation und Visualisierung auf der einen Seite und Experiment und Visualisierung auf der anderen Seite.

Betrachtet man den Stand der Visualisierung in den für diese Arbeit maßgeblichen Bereichen (Kapitel 2.5), so erkennt man, dass zur Visualisierung von Experimentalergebnissen wenig dedizierte Visualisierungslösungen existieren. Es werden vielmehr gängige einfache Verfahren, wie etwa eine Graphendarstellung (Abbildung 49), auch dort verwendet, wo dedizierte Verfahren möglich und sinnvoll wären (Abbildung 50).

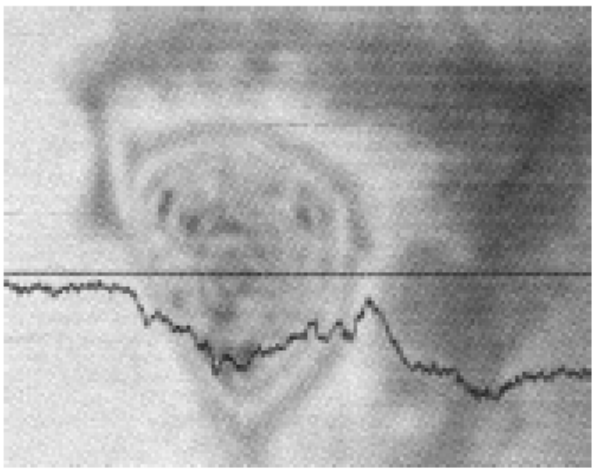


Abbildung 49 Graphendarstellung einer Zelldichtemessung [102]

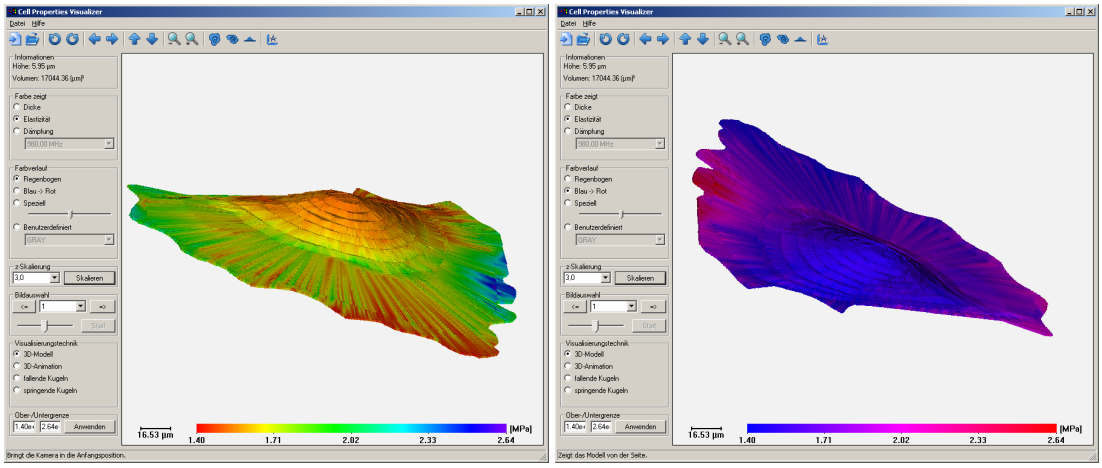


Abbildung 50 Screenshots CellVisualizer

Es hat sich jedoch gezeigt, dass der Einsatz dedizierter Visualisierungssysteme in diesem Bereich bereits durch einen Mangel an für den Anwendungsexperten einfach zu nutzenden Datenvorbereitungs- bzw. Datenaufbereitungswerkzeugen behindert wird [144].

Sofern die Ergebnisse einer Simulationsstudie denen einer konventionellen Experimentalstudie gleichkommen, gilt das eben Beschriebene gleichfalls für den Simulationsfall. Darüber hinaus kommen allerdings simulationsspezifische Anforderungen zum Tragen, die einerseits Anforderungen an die (interaktive) Visualisierung darstellen, andererseits in Anforderungen an die Simulationstechnik und Infrastruktur münden.

Im Folgenden werden entsprechend zunächst Anforderungen an die Simulationstechnik gesammelt, bevor Anforderungen an Interaktivität und Visualisierung betrachtet werden.

3.1. Anforderungen an die Simulation durch die computergestützte Biochemie

Die drei Bereiche, die in dieser Arbeit exemplarisch für die computergestützte Biochemie herangezogen werden, haben neben Gemeinsamkeiten auch Unterschiede, die sich in unterschiedlichen Anforderungen an die Simulation äußern. Die Anforderungen werden im Folgenden untersucht.

3.1.1 Molecular Modeling

Der Bereich des Molecular Modeling ist von den drei betrachteten Bereichen der, in dem traditionell sowohl die *Simulation* als auch die *Visualisierung* am längsten benutzt wird und bearbeitet wurde. Dabei treffen allerdings die beiden „alten“ Paradigmen der Simulation und der Visualisierung aufeinander: monolithische Simulation durch die Aufstellung von Energie- und Bewegungsgleichungssystemen und deren Lösung mit Hilfe von numerischen DGL-Lösern auf der einen Seite und ein von der Simulation getrennter Visualisierungsschritt (siehe Abbildung 51).

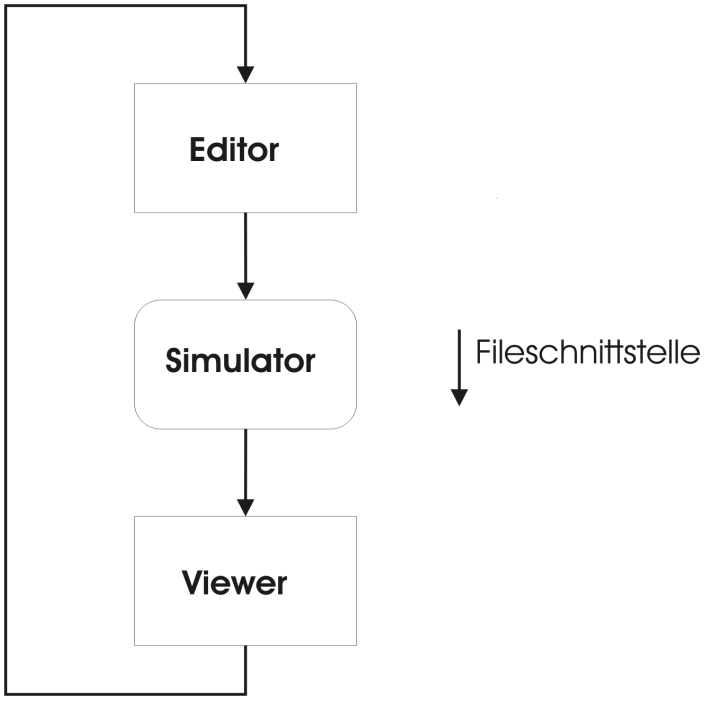


Abbildung 51 Bearbeitungsablauf beim traditionellen Molecular Modeling

Durch diese Konfiguration wird eine **interaktive** Gestaltung der Simulationsstudie stark erschwert: Erst nach Durchführung eines Simulationsdurchgangs, in dem der Zustand des gesamten Systems neu berechnet wird, ist eine Visualisierung der Ergebnisse möglich. Zwar sind die so erzeugten Visualisierungen interaktiv navigierbar, eine direkte, interaktive Rückkopplung zur Simulation ist aber nicht möglich.

Die Anforderung an eine Molecular-Modeling-Applikation mit verbesserten interaktiven Möglichkeiten ist daher die Forderung nach einer direkten Rückkopplung zwischen Editorfunktion, Simulation und Visualisierung.

- *Editorfunktion* beschreibt hier die Funktionalität des eigentlichen Modellierens, also die graphisch interaktive Definition des konkreten Simulationsmodells.
- *Simulation* ist die Berechnung des Systemzustands des gerade editierten Molekülmodells. Die Anforderung an die Simulation ist eine möglichst kurze Berechnungszeit, um interaktives Arbeiten zu ermöglichen.
- *Visualisierung* der Simulationsergebnisse ist die Darstellung des Simulationslaufs. Im Sinne eines kontinuierlichen Editor-Simulations-Visualisierungsprozesses soll die Visualisierung wiederum nahtlos in die nächste Editorphase überleiten.

Für den Bereich der Interaktion im Molecular Modeling wird in weiteren Kapiteln noch einmal speziell unter dem Aspekt der haptischen Interaktion eingegangen.

3.1.2 Systembiologie

Die Situation der Systembiologie differiert insofern ein wenig von der der anderen beiden Betrachtungsfällen, als sie ganz überwiegend durch Simulation als Mittel des

Erkenntnisgewinns geprägt ist. Entsprechend beschäftigt sich eine große Zahl von Publikationen mit dem Einsatz und der Auswahl von Simulationstechniken für die Systembiologie (siehe die folgenden Referenzen zu den jeweiligen Verfahren).

Die Literatur lässt zwei große Gruppen von Forschern erkennen: Die eine Gruppe verteidigt den deterministischen Ansatz, also die Aufstellung und Lösung von Differenzialgleichungssystemen, die zweite Gruppe favorisiert einen stochastischen Ansatz, zumeist über die von Gillespie [144] bzw. von Gibson [145] vorgestellten Methoden. Eine kleine Gruppe versucht, beide Ansätze in so genannten *Hybrid*simulatoren zu verbinden [146] [147]. Bisher hat sich jedoch noch keine hybride Methode durchgesetzt.

Obwohl der Gegenstand der Betrachtung, die biologische Zelle, ohne jeden Zweifel ein räumliches, sprich dreidimensionales Phänomen ist, wird diese Eigenschaft von den gängigen Simulationssystemen weitgehend ignoriert. Vielmehr nimmt die ganz überwiegende Zahl der in der Literatur betrachteten Systeme und Lösungen ein perfekt symmetrisches und gut durchmischtes Reaktionsvolumen an. Diese Annahme ist jedoch in der Natur für keinen relevanten Anwendungsfall gerechtfertigt. Selbst für im Vergleich zu eukariotischen Zellen kleindimensionale Bakterienzellen, die überdies keinen Zellkern haben, ist die Symmetrieannahme nicht gerechtfertigt (siehe Abbildung 52).

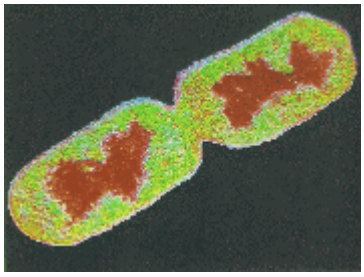


Abbildung 52 e. coli [119]

Werden dreidimensionale intrazelluläre Phänomene mit Methoden der Simulation untersucht, so geschieht dies bisher ausschließlich mit dedizierten Simulationswerkzeugen, die für genau diesen Anwendungsfall erstellt wurden (hier z. B. Zellbewegung des Bakteriums *e. coli* [119], Abbildung 53).

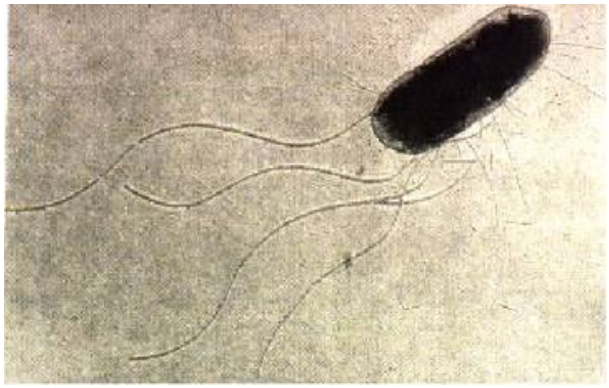


Abbildung 53 e.coli Bewegung [119]

Es soll an dieser Stelle besonders betont werden, dass die Vorgehensweise, statt generischer Lösungen eher nach Insellösungen zu suchen und diese zu entwickeln, nicht auf den dreidimensionalen Charakter des Problems zurückzuführen ist. Vielmehr entspricht diese Vorgehensweise der in diesem Bereich der Forschung in der Biologie vorherrschenden Denkweise: Die detaillierte Aufarbeitung sehr begrenzter Ausschnitte eines Systems wird in der wissenschaftlichen Gemeinschaft (noch) höher bewertet als eine ganzheitliche Betrachtung. Verfechter der Systembiologie [148] werben für eine Änderung dieser Sichtweise hin zu einer Systembetrachung.

Aus dem oben Gesagten ergibt sich eine wesentliche Anforderung: eine Systematik zu finden, die es erlaubt, Simulationen für generische dreidimensionale Phänomene zu erstellen. Darüber hinaus ist ein Weg zu finden, der es ermöglicht, stochastische mit deterministischer Simulation zu verbinden. Diese Verbindung sollte jedoch nicht notwendigerweise mit einer gänzlich neuen Methode erstellt werden. Ziel muss es vielmehr sein, bereits erstellte Modelle zukünftig so weit wie möglich nutzbar zu halten. Dieser Anforderung liegt die Erkenntnis zugrunde, dass ein ganz wesentlicher Teil des Aufwandes, der für eine Simulationsstudie betrieben werden muss, auf die Modellbildung entfällt. Dies geht so weit, dass auf Seiten mancher Forscher der Biologie der Unterschied zwischen Modell und Simulation zugunsten der Modellbildung nicht wahrgenommen wird. Als Resultat ergeben sich Insellösungen, das heißt dedizierte Systeme, in denen Simulation und Modell untrennbar verwoben sind [105].

3.1.3 Zelldifferenzierung

Die Erforschung der Zelldifferenzierung, also die Erforschung des Mechanismus, der es erlaubt, aus einer embryonalen Stammzelle einen gesamten Organismus mit verschiedensten Zelltypen zu erhalten, ist ein gegenwärtig stark beachtetes Gebiet [184]. Für die biologische Forschung ist es hier notwendig, mit embryonalen Stammzellen zu experimentieren und diese damit zu „verbrauchen“. Die ethischen Fragestellungen, die dabei aufgeworfen werden, haben spätestens mit der Diskussion um die Forschung an *menschlichen* embryonalen Stammzellen die breitere Öffentlichkeit erreicht.

Neben dem genannten Stichwort der Stammzellenforschung fallen in diesen Anwendungsbereich auch Untersuchungen zum Verhalten von Zellen in Zellverbänden bis hin zu ihrem Verhalten in Organen. Hierzu gehört auch die Betrachtung des *gesteuerten Zelltods* (Apoptosis) [149].

Der Bereich „Zelldifferenzierung“ hat einige Gemeinsamkeiten zum bereits beschriebenen Bereich der Systembiologie. Während sich die Systembiologie besonders durch den Einsatz von Simulation definiert, gilt das für diesen Bereich nicht. Tatsächlich ist der Einsatz von Simulation für diesen Bereich noch kaum entwickelt. Ähnlich dagegen ist das Modellsubjekt, die biologische Zelle, hier jedoch im Zusammenspiel mit anderen Zellen.

Stößt die praktische Durchführung der Simulationsstudie bereits für eine einzelne Zelle auf rechentechnische Beschränkungen, so ist an die Simulation eines Zellverbandes mit gleichem Modellierungsgrad, wie in der Systembiologie angestrebt, derzeit nicht zu denken. Folgerichtig sind für Zellen in Zellverbänden neue Modelle auf einem höheren Abstraktionsniveau nötig.

Wurde in systembiologischen Modellen oft noch vollständig von den geometrischen Eigenschaften abstrahiert, so ist dies für Zellverbände und Zelldifferenzierung nicht möglich. Bestehende Simulationen beschränken sich allerdings lediglich auf zweidimensionale Geometrien (eine Einschränkung, die den Wert der gewonnenen Erkenntnisse stark relativiert).

Neue spezifische Elemente der Simulation von Zellverbänden und Zelldifferenzierung, die in der Modellbildung berücksichtigt werden müssen, sind neben Aspekten des

Informationsaustauschs, der Kommunikation zwischen den Zellen, wie etwa Zelladhäsion, und des Austauschs von Substanzen insbesondere die Phänomene Zellteilung und Zelltod.

Aus diesen Erkenntnissen ergibt sich eine Reihe von Anforderungen:

- Es ist eine Methodik zu finden, die es erlaubt, den dreidimensionalen Charakter von Zellverbänden zu erfassen.
- Dabei ist insbesondere auf die rechnertechnischen Beschränkungen und den möglichen Einsatz der Verteilung der Simulationsstudie auf Rechnernetzen zu achten.
- Der Besonderheit von Phänomenen wie Zelladhäsion und dem aktiven Austausch von Substanzen muss Rechnung getragen werden.

3.1.4 Zusammenfassung

Aus der Analyse der drei Anwendungsfelder ergibt sich eine Liste von Anforderungen an die Simulation, die sich aus der Anwendung der computergestützten Biochemie ergibt. Diese Anforderungen sind zusammengefasst:

Aus dem Feld des Molecular Modelings ergibt sich die Forderung, die *Interaktivität* durch eine bessere Einbindung der Editor- und Visualisierungsfunktionen in den Simulationszyklus zu unterstützen.

Die Analyse der Systembiologie ergibt weiterhin die Anforderung nach einer Methodik, die es erlaubt, systembiologische Phänomene dreidimensional zu simulieren. Dabei ist darauf zu achten, dass generische Ansätze verfolgt werden, um Insellösungen zu vermeiden. Eine zentrale Forderung ist weiterhin, dass bekannte konventionelle Simulationstechniken und Simulationssysteme miteinander gekoppelt werden können. Dies hat das Ziel, bereits erstellte Modelle und andere Vorarbeiten, etwa spezialisierte Simulationslösungen, weiter verwenden zu können.

Für eine verbesserte Simulation der Zelldifferenzierung sowie von Zellverbänden ist zusätzlich noch die Anforderung nach einer guten rechnertechnischen Skalierbarkeit des Simulationssystems zu beachten, um der hohen Komplexität des Problems Rechnung zu tragen. Aus diesem Anwendungsbereich folgt auch die Forderung, Phänomene wie Zelladhäsion und den aktiven Austausch von Stoffen einheitlich zu adressieren und in das Simulationssystem zu integrieren.

3.2. Anforderungen an Interaktivität und Visualisierung

Während die Anforderungen an die Simulation im vorherigen Kapitel direkt aus den drei betrachteten Bereichen der computergestützten Biochemie entnommen sind, gliedert sich dieses Kapitel in einen allgemeinen Bereich, und erst danach wird jedes der drei Anwendungsfelder gesondert nach Anforderungen analysiert. Es ist zu beachten, dass in diesem Kapitel allgemein von Interaktivität und Visualisierung gesprochen wird; dies ist dabei immer vor dem Hintergrund der computergestützten Biochemie bzw. vor dem Hintergrund der Durchführung einer Simulationsstudie zu sehen.

3.2.1 Allgemeine Anforderungen

In diesem Unterkapitel werden allgemeine Anforderungen an Interaktivität und Visualisierung in Simulationsstudien untersucht. Dabei wird zunächst auf den Erstellungsprozess, also das Editieren und Erstellen des Simulationsmodells in einer computergraphischen Umgebung (3D) eingegangen. Anschließend wird besprochen, welche Auswirkung dies auf die visuelle Darstellung hat, bevor auf die physikalisch-chemischen Randbedingungen eingegangen wird und abschließend die besonderen Anforderungen der Zielgruppe an Interaktion beleuchtet werden.

Eine interaktive Simulationsstudie im Bereich der computergestützten Biochemie enthält immer auch einen Erstellungsprozess innerhalb der Interaktions- bzw. Visualisierungskomponente. Erstellt werden in diesem Zusammenhang die Startkonfiguration des Simulationslaufs und die Konfiguration der Visualisierung. Was dies im jeweiligen Kontext bedeutet, wird weiter unten detailliert.

Wie für alle Visualisierungsaufgaben ist es auch in diesem Fall unumgänglich, die Zielgruppe der Benutzer zu identifizieren. Nutzer einer Anwendung im betrachteten Bereich ist in der Regel ein Experte des Anwendungsgebietes, ein *Domain Expert*. Dies bedeutet, der Nutzer ist nicht notwendigerweise ein Informatikexperte; noch weniger wahrscheinlich ist es, dass der Nutzer über Kenntnisse in computergraphischer Modellierung (3D oder auch 2D) verfügt. Andererseits hat der Nutzer durchaus aus seinem Anwendungswissen heraus eine Vorstellung bezüglich der geometrischen Konfiguration und gegebenenfalls des Aussehens der simulationsrelevanten *Entitäten*.

Es sind demnach zwei Erstellungsebenen zu identifizieren: Auf der einen Ebene sind dies die Eingaben des Nutzers, des *Domain Experts*. Diese Eingaben beziehen sich allerdings auf die Erstellung einer computergraphischen Szene (2D oder 3D). Um das fehlende Wissen und Können des Anwenders im Bereich der Computer-Graphik zu kompensieren, ist zu fordern, dass der Nutzer mit den ihm bekannten Entitäten interagiert. Dies sind für Molecular-Modeling-Anwendungen etwa Atome, Bindungen und Moleküle.

Unter dieser Prämisse ergibt sich eine weitere Autorenrolle, die des *Entitätenautors*. (Dieser Begriff wird später im Rahmen der Konzeptentwicklung durch den Ausdruck „Komponentenautor“ ersetzt werden; die begriffliche Trennung soll ausdrücken, dass das später entwickelte Konzept nicht zwangsläufig das einzige Lösungskonzept ist, sondern die Anforderungen möglicherweise auch mittels eines andern Konzeptes umgesetzt werden könnten.) Dieser Autor erstellt die „Bausteine“, mit deren Hilfe der Endnutzer später seine Simulationsstudie und die Visualisierung ihrer Ergebnisse konfiguriert. In der Verantwortung des Entitätenautors liegt die Ausgestaltung

- der *visuellen Darstellung*,
- des *physikalisch und chemisch korrekten Verhaltens,*
- der zielgruppengerechten, *intuitiven Interaktion*

der Entitäten.

Die allgemeinen Anforderungen an die visuelle Darstellung oder Visualisierung sind die bereits beschriebenen Forderungen nach Expressivität, Effektivität und Angemessenheit. Diese Forderungen müssen für jeden betrachteten Aspekt der computergestützten Biochemie einzeln definiert werden.

Das physikalisch und chemisch korrekte Verhalten ist im Wesentlichen die Domäne der Simulation. An dieser Stelle begründet sich die enge Verzahnung zwischen Visualisierung und Simulation.

Die dritte Anforderung nach einer effektiven Interaktion beschreibt die Art, wie schnell sich ein Nutzer der Zielgruppe in eine gegebene Applikation einarbeiten und selbst produktiv tätig werden kann. Ein Teil dieser Anforderung wird von der Wahl der Visualisierung abgedeckt (Effektivität und Angemessenheit, siehe oben). Darüber hinaus sind jedoch auch Anforderungen an die Metaphern und Geräte zu stellen, mit deren Hilfe der Nutzer mit der Applikation interagiert.

3.2.2 Anforderungen an Interaktivität und Visualisierung durch die computergestützte Biochemie

Die im vorherigen Unterkapitel aufgeworfenen allgemeinen Anforderungen sollen nun für die einzelnen betrachteten Bereiche der computergestützten Biochemie genauer untersucht werden.

3.2.2.1 Molecular Modeling

Ausgangspunkt in Molecular-Modeling-Anwendungen ist, dass die räumliche Gestalt eines Moleküls für seine Funktion, d. h. für seine Eigenschaften, von entscheidender Bedeutung ist. Proteine wirken beispielsweise dadurch, dass sie Bindungsorte für andere Moleküle bereitstellen bzw. an andere Moleküle binden. Diese Bindungsorte weisen sich durch eine charakteristische Geometrie aus. Moleküle in der betrachteten Kategorie sind dabei allerdings in der Regel zu komplex, als dass man eine einfache, intuitive räumliche Vorstellung von ihnen gewinnen könnte. Auch zweidimensionale Visualisierungen sind nicht geeignet, um die inhärent dreidimensionale Geometrie zu beschreiben. Hierdurch sind die spezifischen *Entitäten* dieser Anwendungsklasse definiert:

- physikalische Objekte wie Atome
- Entitäten des zugrunde liegenden chemischen Modells, wie etwa Bindungen
- Felder und ähnliche Effekte, die in der Regel durch Simulation ermittelt und verändert werden.

Der klassische Ansatz des Molecular Modeling erzwingt ein mehrstufiges Arbeiten. Zunächst wird dabei eine visuelle Modellierung vorgenommen. Diese Modellierung ergibt in der Regel keine gültige Konformation des Moleküls, d. h., die modellierten Bindungen, Bindungslängen oder Bindungswinkel können in der Natur nicht vorkommen, oder die Konformationen sind zumindest nicht energieminimal. Daraufhin folgt ein Simulationsschritt, in dem die zwischenatomaren Kräfte berechnet werden und eine wahrscheinliche Konformation gefunden wird. Die Suche nach energetischen Minima ist nur eine mögliche Simulationszielstellung. Daneben können Oberflächeneigenschaften, Felder und die Wechselwirkung mit anderen Molekülen untersucht werden. Das Ergebnis dieses Simulationsschritts wird wieder visuell dargestellt, und der Zyklus beginnt von vorn [12], [13] (siehe Abbildung 51).

Die geforderte Interaktivität wird durch diese starre Vorgehensweise nicht erreicht. Der geforderte Einsatz von Entitäten verlagert nun die Simulation in die jeweilige Entität; in diesem Fall sind dies etwa Atome und submolekulare Strukturen, wie z. B. funktionelle Gruppen. Die Kapselung der Simulation in der Entität erlaubt ein abgestuftes Simulationsmodell. So lassen sich sowohl grobe Modelle, die das Gesamtsystem modellieren, mit Modellen verbinden, die

auf der Ebene der molekularen Mechanik arbeiten. Gleichzeitig ist die Komponente aber auch für ihre visuelle und gegebenenfalls haptische Repräsentation verantwortlich, so dass die beiden klassischen Arbeitsschritte zusammenfallen.

Die Hauptaufgabe des Benutzers ist die räumliche An- und Umordnung von Atomen in Molekülen sowie von Molekülen untereinander. Die auch in der graphischen Darstellung genutzten Metaphern (etwa „Ball and Stick“) entstammen einer historischen Phase, in der physische Modelle aus Holz, Metall etc. zur Visualisierung der 3-D-Strukturen verwendet wurden. Diese Modelle wurden im wörtlichen Sinn durch direkte Manipulationen verändert und konfiguriert. Hieraus ergibt sich die Forderung, diese bekannte und direkte Interaktionsform, also das direkte „Begreifen“ und „Manipulieren“ der Atome und Moleküle, auch für die computergestützte Biochemie zu nutzen.

Der Hinweis auf eine haptische Repräsentation ist ein Vorgriff auf das Konzept, das eine solche Repräsentation einschließt.

3.2.2.2 Systembiologie

Zur Betrachtung der Anforderung, die die Systembiologie an die Visualisierung und Interaktion stellt, sollen die beiden Gebiete, Visualisierung und Interaktion, zunächst getrennt betrachtet werden.

3.2.2.2.1 Visualisierung

Die Systembiologie hält zwei traditionelle Bereiche vor, für die Visualisierung eine Rolle spielt: Zum einen bedarf es einer Visualisierung des Simulationsmodells, zum anderen muss das Ergebnis der Simulationsstudie visualisiert werden.

Die meisten der bekannten Simulationsmodelle bestehen bisher aus der Formulierung von chemischen Reaktionsnetzwerken. Da Reaktionsnetzwerke in erster Regel *Netzwerke* sind, lassen sich alle Erkenntnisse der Visualisierung von Netzwerken [150] auch für diesen Anwendungsfall nutzen. Dabei wird jedoch nicht beachtet, dass das chemische Reaktionsnetzwerk in Wirklichkeit nur ein Teil des gesamten Modells ist: Wie bereits im Kapitel *Anforderungen an die Simulation* beschrieben, ignorieren viele der bestehenden Modelle die dreidimensionale Natur des Modellierungssubjekts „Zelle“. Dass es noch kaum angemessene Modelle gibt, liegt auch am Fehlen adäquater Modellierungsansätze. Das Fehlen dieser Modellierungsansätze ist, nach meiner Auffassung auch durch das Fehlen entsprechender Modellierungstools für den 3-D-Fall begründet. Hieraus ergibt sich die Anforderung an die Visualisierung in dieser Arbeit. Die zu betrachtenden *Entitäten* im Sinne der vorausgegangenen Überlegungen sind hier die in einer Zelle vorkommenden Organellen, wie Ribosomen oder der Zellkern.

Während das Simulationsmodell, wie beschrieben, in der Regel als Netzwerkvisualisierung dargestellt wurde, lassen sich die Simulationsergebnisse hier nicht so einfach visualisieren. Die bisherige Graphendarstellung ermöglicht keine einfache Verbindung zwischen Modell- und Ergebnisvisualisierung. Hieraus ergeben sich zwei weitere Anforderungen der Systembiologie an die Visualisierung: Zum einen muss untersucht werden, inwieweit Ergebnisvisualisierung und Modellvisualisierung im Rahmen einer Netzwerkvisualisierung enger verzahnt werden können, zum anderen müssen die dreidimensionalen Aspekte der Simulation, etwa die Diffusion, auch angemessen in einer 3-D-Visualisierungsumgebung dargestellt werden. Diese Umgebung und die mit ihr verbundenen Entitäten entsprechen der oben zur Modellvisualisierung beschriebenen Umgebung.

3.2.2.2.2 Interaktivität

Die Anforderungen an die Interaktivität, die sich aus den Notwendigkeiten der Systembiologie ergeben, sind eng verknüpft mit den Forderungen, die sich aus der Visualisierung ergeben. Oben wurden die Bereiche „Modellvisualisierung“ und „Visualisierung der Simulationsergebnisse“ angesprochen. Für eine interaktive Modellbildung bedarf es entsprechender Modellierungstools. Diese Werkzeuge existieren im Fall der 2-D-Reaktionsnetzwerke, sie fehlen jedoch im dreidimensionalen Fall. Leider existieren bisher noch keine überzeugenden Ansätze der 3-D-Modellbildung auf der Seite der biologischen Modellierung [156], so dass sich die Anforderungen im Rahmen dieser Arbeit zunächst nur mit der interaktiven Erstellung der 3-D-Simulationsumgebung beschäftigen. Benötigt wird hier ein Autoren- und 3-D-Modellierungswerkzeug für biologische Zellen. (3-D-Modellierung meint in diesem Fall die rein computergraphische Modellierung; leider wird in beiden Bereichen, Computergraphik wie auch Simulation, der gleiche Begriff „Modell“ benutzt). Ein solches Tool hat darüber hinaus Bedeutung für die interaktive Konfiguration der Simulationsläufe. Während es im traditionellen Fall ausreicht, allein die Stoffmengen zu spezifizieren und damit den Simulationslauf zu konfigurieren, reicht dies im dreidimensionalen Fall nicht aus. In diesem Fall muss nicht nur spezifiziert werden, *welche Menge* eines bestimmten Stoffes vorhanden ist, sondern auch *wo* in der Zelle dieser Stoff vorhanden ist. Hierzu bedarf es solcher Werkzeuge, die geeignet sind, nicht nur die oben genannten Organellen zu konfigurieren, sondern auch die für die Reaktionen relevanten Substanzen, Proteine und andere Chemikalien interaktiv im Raum zu positionieren. Die Veränderung von Stoffmengen und Position wird durch die Simulation bestimmt. Es ist folgerichtig, die Ergebnisse der Simulation in der gleichen Umgebung zu visualisieren, die auch für die Konfiguration genutzt wurde.

3.2.2.3 Zelldifferenzierung

Die Herausforderungen der Simulation von Zelldifferenzierung und Zellverbänden gehen einher mit entsprechenden Anforderungen an eine interaktive Visualisierung. Ganz ähnlich zur Situation in der Systembiologie bedarf es im hier betrachteten Gebiet zweier interaktiver Visualisierungs- und Editorsysteme: Zum einen wird ein für den Biologen leicht zu bedienender, da graphisch orientierter Modelleditor benötigt. Dieser Editor ist, was die Stoffumsetzungen etc. betrifft, dem der Systembiologie sehr ähnlich; die verwendeten Modelle unterscheiden sich allerdings. Zum anderen wird ein Experiment-Editor gebraucht, der die Erstellung von Szenarien erlauben soll, um gezielte Experimente an den virtuellen Zellen durchführen zu können. So könnte beispielsweise die initiale molekulare Ausstattung einer Zelle verändert und die Auswirkungen auf die Zelldifferenzierung simuliert werden.

Die Simulationsergebnisse sollen ebenfalls interaktiv visualisiert werden, um ein Eingreifen in die Simulation zu ermöglichen. Dies schließt sowohl die Navigation in der Szene, die Anwendung von Filtern (Anzeige von Zellen mit bestimmtem Status) und die Steuerung von äußeren Einflüssen (Zugabe von Substanzen in das Medium) ein.

Bei all diesen graphischen Systemen ist zu beachten, dass der biologische Forscher bereits eine aus der Mikroskopie entwickelte Vorstellung für die Darstellung von Zellverbänden hat (siehe Abbildung 54). Die künstlich erzeugten Bilder müssen sich mit diesen realen Aufnahmen vergleichen lassen und zusätzlich weitergehende Informationen visualisieren, wie sie nur durch Simulation erhältlich sind, etwa über den nicht mit dem Auge wahrnehmbaren inneren Zustand der Zelle.

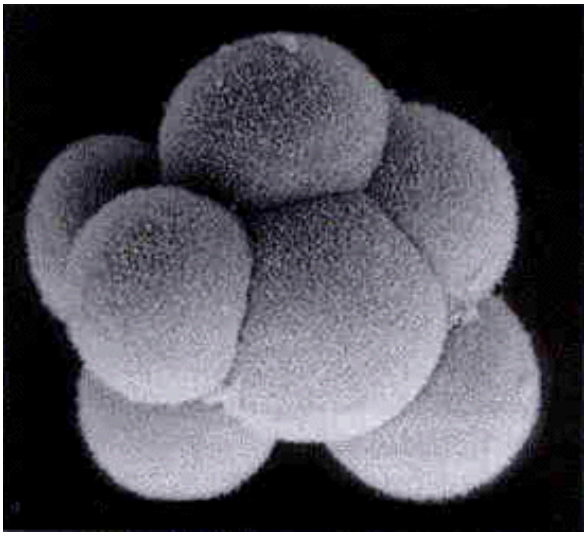


Abbildung 54 Zellhaufen

3.3. Anforderungsanalyse jenseits der computergestützten Biochemie

Zwar sind die bisher vorgebrachten Anforderungen aus den Bedürfnissen der computergestützten Biochemie hervorgegangen, jedoch zeigt sich, dass Bereiche jenseits dieses Feldes ganz ähnliche Anforderungen an Simulation, Interaktivität und Visualisierung stellen können. Im Folgenden soll dies anhand einger konkreter Beispiele unterlegt werden.

Jedes der ausgewählten Anwendungsfelder – *Kontrolle chemischer Prozesse im industriellen Kontext*, *Training für Notfallsituationen* und *Einrichtungsplanung*, im Zusammenhang interaktiver Simulation und Visualisierung – wäre für sich genommen wichtig und breit genug, eine eigene Arbeit in diesem Umfang zu rechtfertigen. An dieser Stelle soll daher keine umfassende Bestandsaufnahme und Anforderungsanalyse durchgeführt werden, vielmehr sollen Parallelen zwischen diesen Gebieten und dem Hauptgebiet, der computergestützten Biochemie, aufgezeigt werden.

3.3.1 Kontrolle chemischer Prozesse im industriellen Kontext

Chemische Prozesse im industriellen Kontext bestehen in der Regel aus einer Reaktionskaskade, an deren Anfang die in den Prozess einzubringenden Ausgangsstoffe stehen und am Ende, gegebenenfalls über eine Reihe von Zwischenprodukten, die gewünschten Endprodukte. Dabei sind nicht nur die Tatsache der Reaktion an sich, sondern auch Ausbeute und Qualität der Resultate von wirtschaftlicher Bedeutung.

Welche Rolle spielen hierbei Simulation und Visualisierung? Diese Frage lässt sich anhand eines konkreten Beispiels verdeutlichen: Zur industriellen Gewinnung von hochdichtem Polyethylen (High Density Polyethylen, HD PE, einem häufig im Alltag verwendeten Kunststoff) wird heute ein von Karl Ziegler im Jahr 1953 entwickeltes Verfahren, die Ziegler-Polymerisation [151], im industriellen Maßstab verwendet (siehe Abbildung 55).

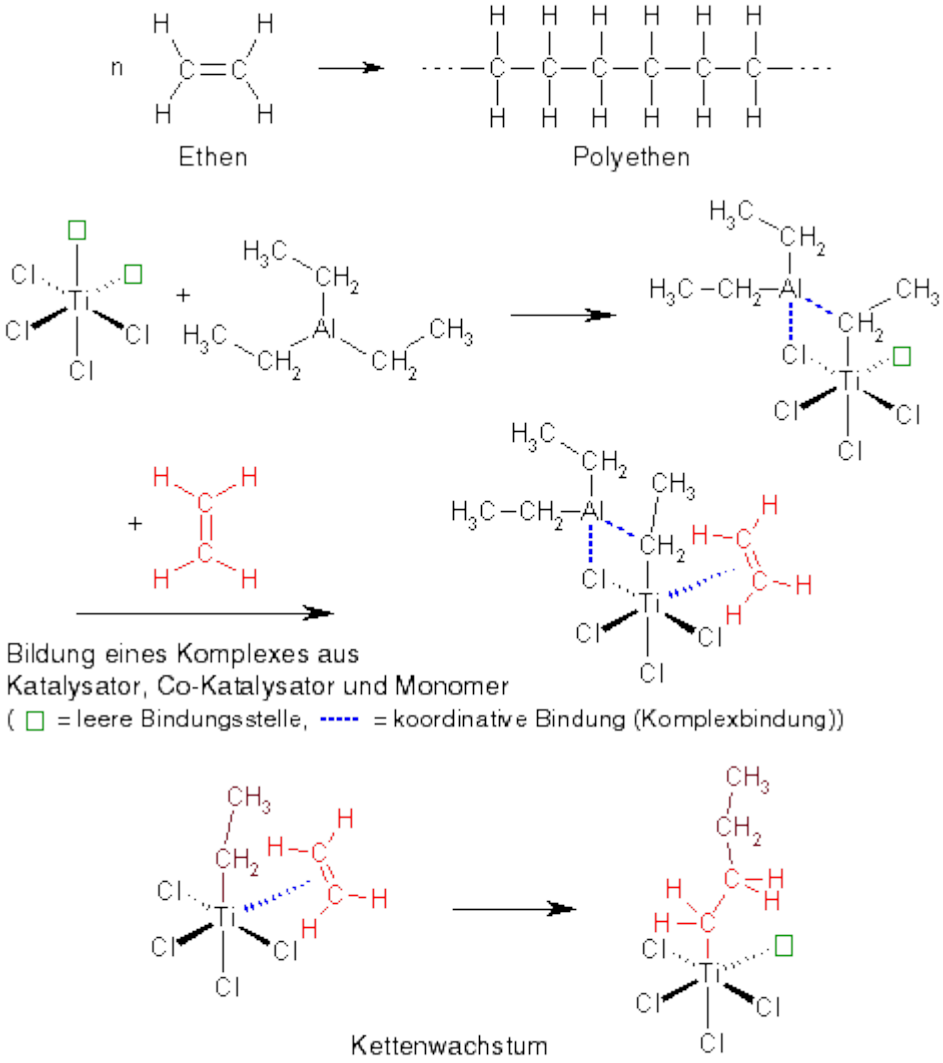


Abbildung 55 Oben: Prinzip d. Polymerisation, Mitte: Katalysatoren, Unten: Polymerisation mittels Ziegler-Verfahren [151]

In diesem Verfahren wird unter dem Einfluss metallorganischer Verbindungen aus den Ausgangsstoffen Ethylen und Wasserstoff Polyethylen synthetisiert. Für die industrielle Weiterverarbeitung des Materials ist es dabei von größter Bedeutung, signifikante Qualitätsparameter wie *Dichte* und *Melt-Flow-Rate* (dt. Schmelzindex) innerhalb vorher bestimmter Grenzwerte zu halten. Die Bestimmung der Qualitätsparameter kann allerdings nur im Labor erfolgen und dauert zu lang, um eine wirkungsvolle Regelung der Zuflüsse von Eingangsstoffen und Katalysatoren zu ermöglichen. Um den Prozess dennoch geregelt „fahren" zu können, werden die Qualitätsparameter prozessbegleitend simuliert.

Die Simulation ist *interaktiv* in der Hinsicht, dass Veränderungen, die an den Stellgrößen vorgenommen werden, direkt an den Simulator weitergegeben und dort verarbeitet werden

können. Auch ohne eine direkte Anbindung an den tatsächlichen chemischen Prozess ist das Simulationssystem, Modell und Simulator, von Interesse: Die mögliche (und auch tatsächliche) Nutzung dieses Systems reicht vom Einsatz in Trainings- und Ausbildungsszenarien über den Einsatz im Marketing Support, etwa bei Messen, bis zur Offline-Prozessanalyse.

Für diesen Einsatz ergibt sich eine Reihe von Anforderungen an die Visualisierung und Interaktion. Vergleicht man die Situation hier mit der in der computergestützten Biochemie, so erkennt man, dass die Entitäten hier das Reaktorgefäß sowie die Ventile, die den Einlass der verschiedenen an den Reaktionen beteiligten Chemikalien regeln, sind. Auch die Chemikalien selbst sind natürlich Entitäten der Simulation. (In Analogie zum industriellen Prozess soll der Nutzer nicht mit den Chemikalien direkt interagieren, sondern mit den Stellmechanismen, wie etwa mit Ventilen.) (siehe Abbildung 56 in Vorgriff auf die später realisierte Lösung)

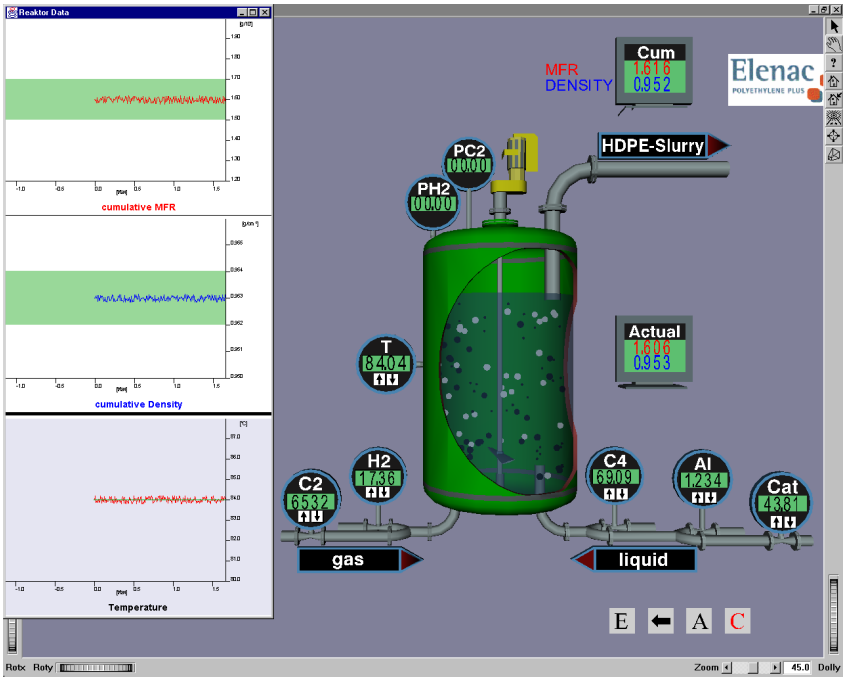


Abbildung 56 Chemischer Reaktor

Diesen Interaktionsanforderungen stehen darüber hinaus Anforderungen an die Visualisierung gegenüber. An die Visualisierung werden zwei grundsätzlich Anforderungen gestellt: Zum einen könnte es Ziel der Visualisierung sein, den Polymerisationsprozess als solchen zu visualisieren, um so das Verständnis für diesen Vorgang zu erhöhen. In jedem Fall muss es jedoch Ziel der Visualisierung sein, den zeitlichen Verlauf der Stellgrößen und der Qualitätsparameter zu darzustellen.

3.3.2 Training für Notfallsituationen

Der weite Bereich des Trainings für Notfallsituationen reicht vom Erste-Hilfe-Training über die Ausbildung von Rettungspersonal bis zur Planung des Katastrophenschutzes für Katastrophen im internationalen Maßstab. Der spezielle Bereich des Trainings für Notfallsituationen, der hier betrachtet werden soll, beschäftigt sich mit dem Training des Kommunikationsverhaltens der handelnden Personen in Organisationen, die normalerweise nicht direkt mit Notfallsituationen konfrontiert sind. Es hat sich gezeigt, dass das Training für Notfallsituation in den meisten

Fällen auf das Einüben operativer Fertigkeiten ausgerichtet ist. So werden z. B. die Feuerwehrkräfte, die im Notfall einen Tunnelbrand in einer Metro bekämpfen, regelmäßig entsprechend trainiert. Dagegen kann das Kontrollraumpersonal, wie etwa der Schichtleiter, bisher nicht ausreichend trainiert werden. Der Grund hierfür liegt vor allem in logistischen Schwierigkeiten, denn für das Training der Führungskräfte muss in der Regel eine große Anzahl von Personen eingesetzt und koordiniert werden. Hinzu kommt, dass üblicherweise in schweren Notfällen Mitglieder verschiedenster Organisationen tangiert werden (Behörden, externe Rettungsdienste etc.). All diese Organisationen müssten in ein solches Training eingebunden werden.

Durch den Einsatz von Simulation und Visualisierung können einige der oben genannten Probleme, die derzeit einem intensiveren Training entgegenstehen, gelöst werden. Hieraus ergeben sich Anforderungen an die Simulation und die Visualisierung. Gegenstand der Simulation der Trainingsanwendungen sind zunächst die Objekte der realen Umwelt, die für die Notfallsituation eine Rolle spielen, etwa ein Feuer, das in einer Industrieanlage ausbricht. Derartige Simulationen sind die Domäne der „klassischen“ Sichtsysteme, wie etwa Flugsimulatoren oder Leitwartensimulationen in Kraftwerken [152]. Von viel größerem Interesse im Kontext des Kommunikationsverhaltens sind allerdings die Aktionen und Reaktionen der agierenden Personen. Sollen die logistischen Probleme, die durch das Zusammenholen einer großen Anzahl von Trainingsteilnehmern entstehen, vermindert werden, so ist das Verhalten einiger Personen zu simulieren. Die Entitäten der Simulation sind hier also, ebenso wie die Entitäten der Visualisierung, die agierenden Menschen in der entsprechenden Umgebung.

3.3.3 Einrichtungsplanung

Die Planung der Einrichtungen von Wohn- und Geschäftsräumen ist ein häufig auch von Laien genutztes Anwendungsfeld der Computergraphik. Hierbei muss zunächst vom Anwender ein dreidimensionaler Architekturplan erstellt und editiert werden. Die eigentliche Visualisierungsaufgabe ist dann die Ausgestaltung der spezifizierten virtuellen Räume mit Wandgestaltungen (Tapeten etc.) Lichtquellen und vor allem Möbeln.

Die betrachteten Gegenstände und Objekte müssen also im Raum positioniert werden, dabei spielen neben den geometrischen Abmessungen vor allem ästhetische und semantische Aspekte eine Rolle (siehe Abbildung Abbildung 57). Semantische Aspekte meint hier vor allem funktionelle Überlegungen. Demnach sollte man Schränke nicht so vor Türen stellen, dass diese sich nicht mehr öffnen lassen etc. Darüber hinaus sind Möbel in der Einrichtungsplanung vor allem auch Waren und kommerzielle Produkte, die Attribute wie „Verfügbarkeit“, „Preis“, „Lieferzeit“, aber auch „Produktfamilie“ etc. haben können.

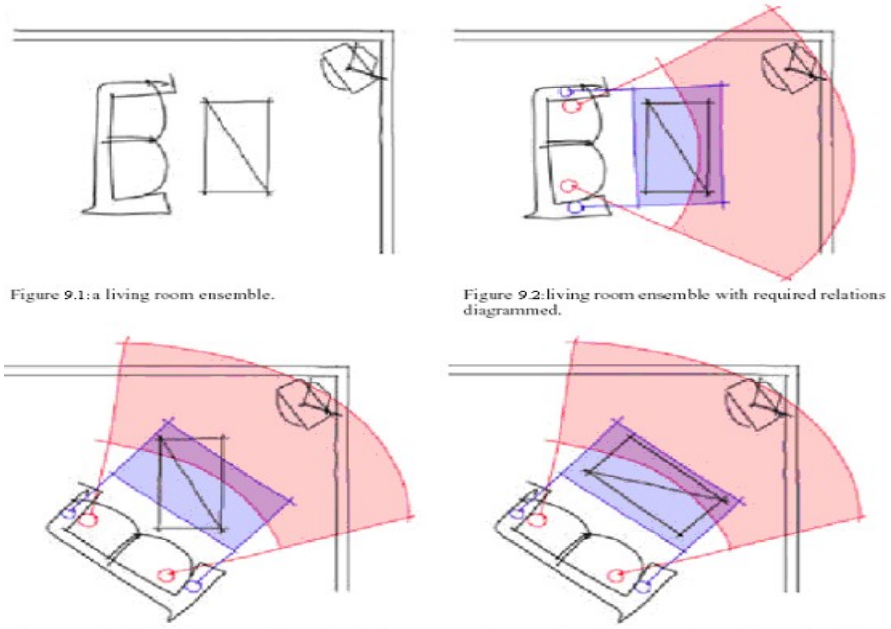


Abbildung 57 Semantische Constraints Sofa [IST-2001-34159]

Ähnliche Aufgaben und Überlegungen lassen sich auch für andere Bereiche finden, in denen virtuelle Objekte in einem kommerziellen und semantischen Kontext konfiguriert werden müssen, etwa in interaktiven Produktkatalogen, in interaktiven Produkthandbüchern etc. (siehe Abbildung 58).

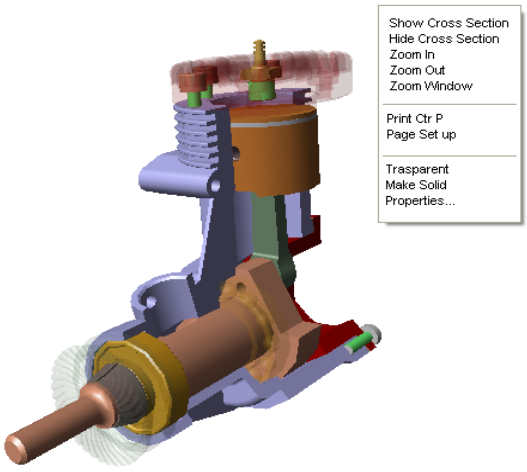


Abbildung 58 Explosionszeichnung [Democenter]

Tatsächlich handelt es sich bei der eben beschriebenen Aufgabe um eine interaktive Simulation und Visualisierung: Die Aufgabe der Visualisierung ist es, einen Eindruck von den so noch nicht existierenden Räumen zu gewinnen. Der Simulationsaspekt ist hierbei das „richtige Verhalten“ der Objekte bezüglich ihrer physikalischen Eigenschaften, etwa Positionierbarkeit, Beleuchtbarkeit bzw. Beleuchtung anderer Objekte etc. Hinzu kommen die beschriebenen „kommerziellen“ Attribute.

Um dieses Simulationsziel zu erreichen, müssen daher die räumlichen 3-D-Modelle eines Produktes mit semantischen Informationen zu diesem Produkt verbunden werden. Semantische Informationen umfassen in diesem Fall neben den oben beschriebenen Attributen im Übrigen auch Informationen wie: "Objekt X lässt sich besonders gut mit Objekt Y kombinieren".

Die Identifikation von Entitäten fällt in diesem Bereich besonders leicht, denn wir haben es hier in erster Linie mit Objekten der realen Welt zu tun. Weniger offensichtlich, aber leicht einsehbar ist es, auch solche „Objekte“, die in der realen Welt nicht als abtrennbare Gegenstände existieren, als Entitäten zu verstehen, wenn sie in unserer Begriffswelt abgeschlossene, logische, mit semantischen Inhalten belegte Objekte sind. So verbinden wir durchaus eine abgeschlossene Vorstellung mit den Begriffen „Wohnzimmer“ oder „Schlafzimmer“.

Die Zweiteilung der Autorenrollen ist in diesem Beispiel ebenfalls leicht erkennbar: Der Benutzer des Visualisierungs- und Simulationssystems konfiguriert Objekte, setzt sie in Beziehung etc. Die zu diesem Zweck genutzten Entitäten sind von entsprechenden Autoren der Entität bereits vorbereitet. Es ist durchaus wahrscheinlich, dass es sich bei dem Entitätenautor nicht um eine einzelne Person handelt; ebenso ist offen gelassen, wie das „richtige Verhalten“ simuliert wird. Diese Details sollen dem Endanwender verborgen bleiben.

3.4. Zusammenfassung

Aus der Analyse der Anforderungen der computergestützten Biochemie und dreier weiterer exemplarisch betrachteter Anwendungsgebiete ergeben sich in der Zusammenfassung drei wesentliche Anforderungen:

Zu finden sind

- ein *Simulationskonzept*, welches es erlaubt, die dreidimensionale Natur der betrachteten Zielsysteme, Moleküle, Zellen, Zellverbände (aber auch Elemente chemischer Prozesstechnik, menschliches Verhalten in Notfallsituationen und virtuelle Möbel) zu beschreiben. Dabei ist nicht nach neuen Simulationstechniken zu suchen, sondern vielmehr nach einem Konzept, das bekannte und bewährte Verfahren verbindet;
- ein *Visualisierungskonzept*, das es erlaubt, *Entitäten* zu identifizieren und diese ihren „Eigenschaften“ entsprechend (siehe Simulation) zu modellieren. Es ist zu beachten, dass die Abbildung in der Realität existierender Objekte auf virtuelle 3-D-Objekte leicht fällt, allerdings muss auch für solche Entitäten eine Abbildung gefunden werden, die in der Realität keine geometrische Verkörperung haben;
- ein *Interaktionskonzept*, das einen interaktiven sowie einen in Abhängigkeit von der jeweiligen Zielgruppe intuitiven Zugang zu den Visualisierungs- und Simulationsentitäten erlaubt.

4. Konzept

In den vorhergehenden Kapiteln wurde eine Reihe von Problemen und Fragestellungen thematisiert, die die computergestützte Biochemie derzeit beschäftigt. In diesem Kapitel wird ein Konzept vorgestellt, das vor dem Hintergrund der beschriebenen Grundlagen für diese Probleme Antworten formuliert.

Analog zur Zweitteilung der Anforderungsanalyse in *Anforderungen an die Simulation* und *Anforderungen an die Interaktivität und Visualisierung* wird auch hier zunächst eine Konzept für die Simulation, ein weiteres für die Interaktion und Visualisierung und anschließend die Integration der Konzepte vorgestellt. Selbstverständlich sind die beiden Konzepte nicht zufällig oder unabhängig voneinander entstanden, sondern sie sind vielmehr Ergebnis eines Vergleichs der Anforderungen aus beiden Bereichen sowie der Einbeziehung bereits existierender Lösungen. Entsprechend soll hier zunächst der Grundgedanke der Lösungskonzepte vorgestellt werden.

Als Grundkonzept des Lösungsansatzes wird ein *komponentenbasiertes* Konzept vorgeschlagen. Als Komponenten werden Software-Einheiten angesehen, die sich durch eine festgelegte Schnittstelle definieren und damit angesprochen und benutzt werden können. Sie können als eine Weiterentwicklung der objektorientierten Konzepte in der Hinsicht angesehen werden, dass sie die Wiederverwendbarkeit weiter erhöhen wollen [11]. In den letzten Jahren wurden mehrere Arbeiten darüber veröffentlicht, wie diese Konzepte für die 3-D-Computergraphik genutzt werden können, wobei die Motivation für diese Arbeiten sehr unterschiedlich sind und somit auch die Betrachtungsweise nicht einheitlich ist.

Für den effektiven Einsatz von Komponenten bedarf es eines *Frameworks*, in dessen Rahmen die Komponenten konfiguriert und verwaltet werden [10], [11]. Frameworks sind Skelettanwendungen, die als Adapter zur jeweiligen Hardwarekonfiguration und dem Anwendungshintergrund dienen.

Es wird gezeigt, unter welchen Vorraussetzungen sich dieses Konzept für die interaktive Simulation und Visualisierung in der computergestützten Biochemie einsetzen lässt.

4.1. Konzept-Simulation

Aus der Anforderungsanalyse der Situation der Simulation in der computergestützten Biochemie ergeben sich zwei Beobachtungen: Zum einen stellen sich simulationstechnische Fragen, etwa bezüglich der Auswahl des geeigneten Verfahren und der Handhabung des dreidimensionalen Charakters des Modellsystems. Zum anderen zeigt sich, dass der Ansatz der ganzheitlichen systematischen Betrachtung im Konflikt zur traditionellen Sichtweise der Forscher steht.

Zur Lösung der aufgeworfenen Probleme wird ein „Baukastensystem“ vorgeschlagen. Die in der Vorüberlegung genannten Komponenten sind hier „Bausteine“ eines Gesamtsystems. Die Baussteine, die Komponenten, haben dabei verschiedene Rollen, die sie im Zusammenspiel annehmen und ausfüllen. Die Auswahl des geeigneten Simulationsverfahrens, z. B. die Auswahl eines stochastischen Verfahrens gegenüber einem deterministischen Verfahrens, wird so zur Auswahl eines Bausteins, wobei gezeigt werden wird, wie z. B. mit Hilfe von Regeln die Auswahl auch automatisiert werden kann. Gleichzeitig kann man „Baussteine“ auch im räumlichen Sinne verstehen als Dinge, die in einem dreidimensionalen Kontext einen bestimmten Raum einnehmen und die, in ihrer Gesamtheit betrachtet, den Beobachtungsraum der Simulation ausfüllen. Schließlich korrespondieren die Bausteine auch mit den durch die

Anforderung an die Interaktion definierten *Entitäten*. So lässt sich eine Entität oder eine definierte Gruppe von Entitäten häufig direkt durch eine Simulationskomponente abbilden.

Als Beispiel seien hier in Abbildung 59 die Möglichkeiten des Baukastenprinzips anhand des in der Systembiologie üblichen Modellsubjekts „Zelle“ verdeutlicht.

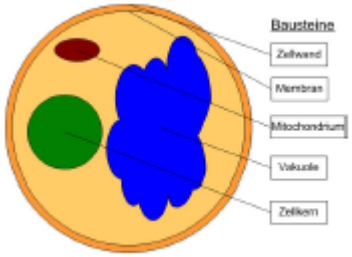


Abbildung 59 Komponenten in der Systembiologie

Sieht man die verschiedenen Bausteine also als Simulatoren, so lässt sich das Konzept als Kopplung verschiedener und gegebenenfalls verschiedenartiger Simulatoren erklären. Zu diesem Zweck wird die Kapselung der verschiedenen Simulatoren voneinander vorgeschlagen. Diese Kapselung und damit die Entkopplung der Simulatoren voneinander übernimmt dabei jeweils ein intelligenter Software-Agent. Zusammengesetzt ergibt das gesamte System also ein Multi-Agentensystem. Durch diese Kapselung ist es möglich, bereits vorhandene Simulationssysteme weiterzuverwenden.

4.1.1 Kopplung heterogener Simulatoren durch Agenten

Das Konzept der Kopplung heterogener Simulatoren durch Agenten ist abgestimmt auf das Interaktionskonzept. Wie später ausgeführt wird, lassen sich über die Agentenschicht dedizierte, auf den jeweiligen Simulator abgestimmte Interaktionsschemata realisieren.

Die Nutzung von Agentensystemen als Simulationssystem wurde bereits beschrieben. In den überwiegenden Fällen dieser Art findet dabei die Simulation als Teil des Verhaltens des Agenten statt. Es ist wichtig, darauf hinzuweisen, dass für dieses Konzept **nicht** davon ausgegangen wird, die verschiedenen Simulationstechniken *innerhalb* des Agenten zu implementieren. Vielmehr soll der Agent als Adapter zwischen dem eigentlichen Simulationssystem und dem gesamten Framework fungieren (siehe Bild Abbildung 60).

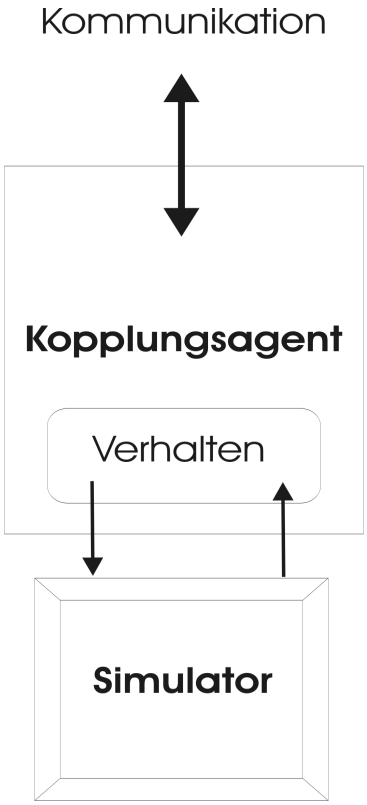


Abbildung 60 Simulation-Agenten Kopplung (abstrakt)

Auch für diesen Einsatz von Agentensystemen gibt es Präzedenzfälle, diese sind jedoch im Vergleich zu den vorher beschriebenen Simulationssystemen selten. Als Beispiel für die Kopplung von Simulatoren seien hier [153] und, allgemeiner, [154] genannt.

Das erste Beispiel, [153], bezieht sich auf die Simulation von Wasserfahrzeugen unter Berücksichtigung von Umwelteinflüssen. Allgemeiner wird die Einbindung von Drittsoftware in Agentensystemen von der FIPA in einem experimentellen „Standard“ behandelt [154]. Obwohl die FIPA ein allgemeines Szenario anbieten will, zeigt sich bei genauerer Betrachtung schnell, dass die FIPA bei der Anbindung von Drittsoftware vor allem an „Services“ denkt, wie etwa die Anbindung einer Datenbank (siehe hierzu die Abbildung 61). Dieses Paradigma ist allerdings nicht sehr gut geeignet für die Anwendung im Zusammenhang der Simulation.

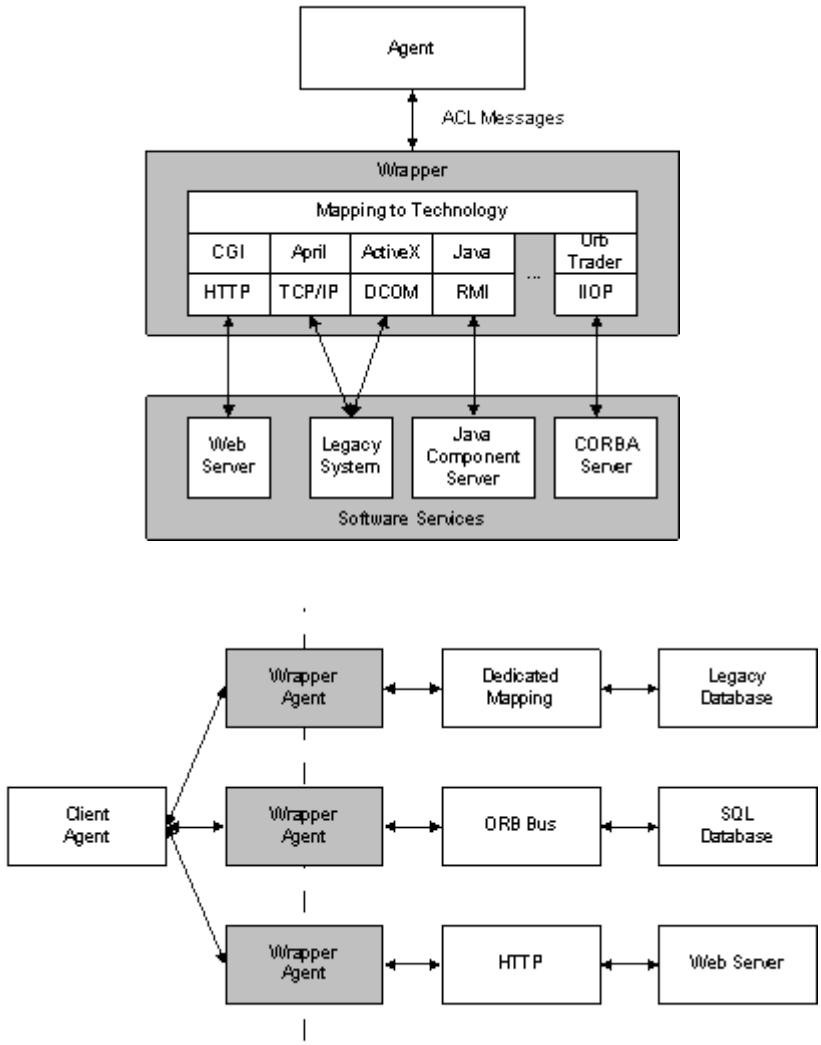


Abbildung 61 FIPA-Wrapper-Agent [34]

Die Idee, Multi-Agentensysteme für die Forschung in den biologischen Wissenschaften einzusetzen, wurde bis jetzt nur in wenigen Anwendungen verwirklicht, so z. B. in GeneAgent [155] (siehe Abbildung 62).

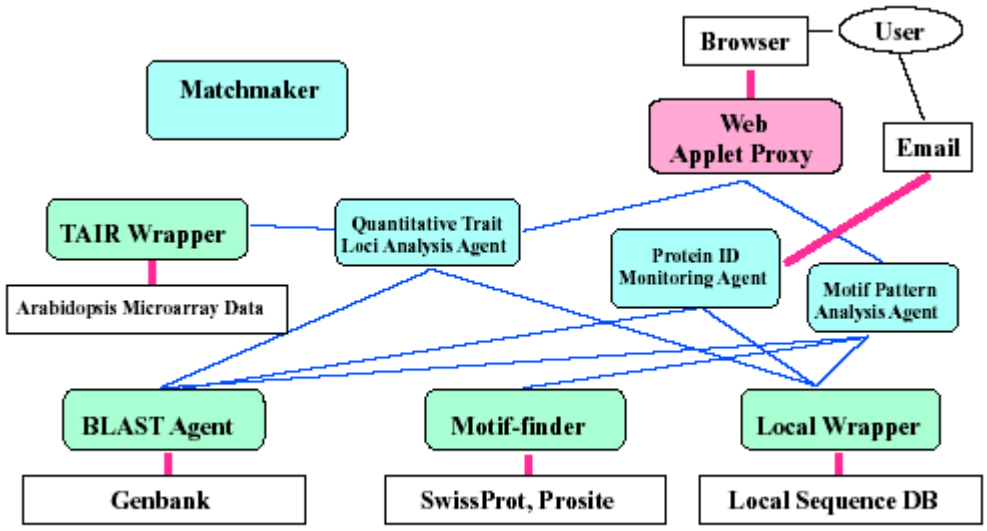


Abbildung 62 GeneAgent-Architekture [155]

Diese Anwendung des Multi-Agenten-Ansatzes entspricht eher dem FIPA-Denkmuster des *Service Providers*.

In dem hier vorgestellten Ansatz wird dagegen nicht von einem Agenten als Service Provider ausgegangen, vielmehr ist der Agent Teil der Simulation; sein Verhalten wird durch die Simulation, die er kapselt, bestimmt. Die in Abbildung 60 beschriebene allgemeine Architektur zeigt den Agenten im Zusammenspiel mit den anderen Agenten und dem Simulator. In Abbildung 63 wird noch einmal die Situation des Agenten genauer gezeichnet.

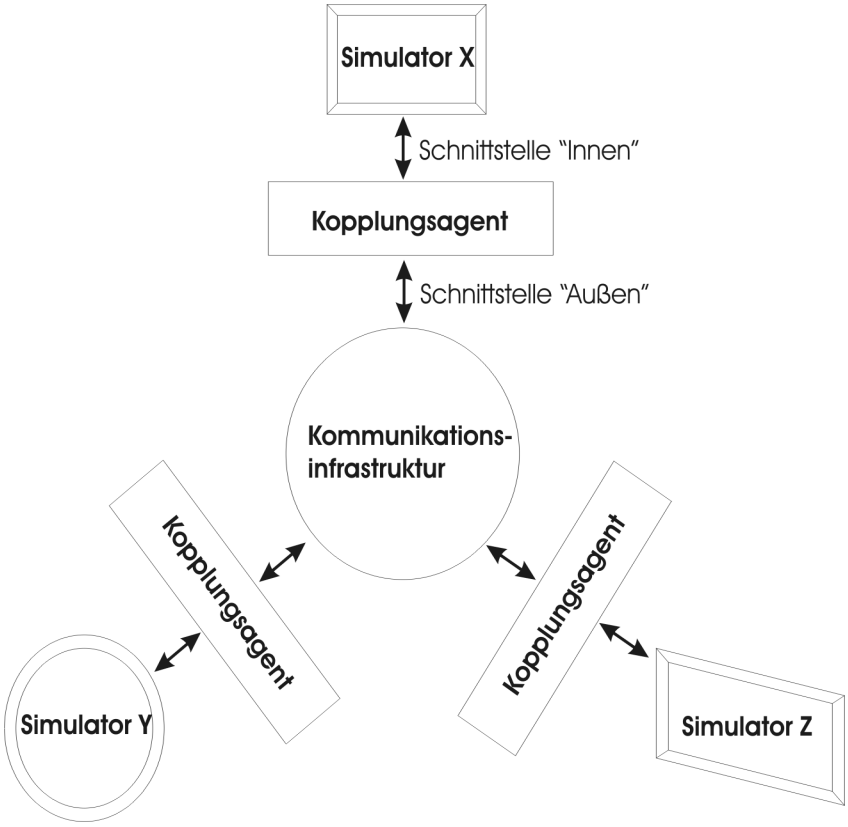


Abbildung 63 Simulation Agenten-Kopplung

Es zeigt sich, dass der *Kopplungsagent* durch zwei Schnittstellen beschrieben werden kann: eine Schnittstelle nach „innen“, zur Simulation, sowie eine Schnittstelle nach „außen“, zu den anderen Agenten.

Die Schnittstelle nach „innen“ ist verantwortlich für:

- Konfiguration der Randbedingungen der Simulation
- Steuerung des Simulationslaufs
- Visualisierung der Ergebnisse.

Die Schnittstelle nach außen zeichnet für die Kommunikation zwischen den Agenten verantwortlich.

Aus Sicht der computergestützten Biochemie ergeben sich zwei Varianten für die Ausgestaltung der Gesamtarchitektur. Die Varianten sind abhängig von der Art der konkreten Anwendung, genauer von den jeweils bereits benutzten Simulatoren, und beeinflussen die *Granularität* des Agentensystems.

Eine *grobe Granularität* bedeutet wenige „große“ Agenten. „Groß“ meint hier, dass entweder mehrere Entitäten in einem Simulator abgebildet werden oder dass große Bereiche des Beobachtungsraums des Modells von einem Simulator abgedeckt werden. Diese Variante findet z. B. bei der Verschaltung von Simulationssystemen wie Matlab [54], Cerius2 [96] oder etwa MolCAD [97] Anwendung.

Eine *feine Granularität* zeichnet sich durch eine geringere Komplexität des einzelnen Agenten im Kontext des gesamten Systems aus; dagegen steigt die Anzahl der Agenten. Der Zusammenschluss kleiner spezialisierter Simulatoren, wie etwa für das Molecular Modeling oder die Zelldifferenzierung, ist ein Beispiel für eine feine Granularität.

Für die spätere Umsetzung dieses Konzeptes ist die Frage zu beantworten, ob es für die zugewiesene Rolle des Kopplungsagenten einer besonderen Agentenimplementierung bedarf. Diese Frage ist mit „Nein“ zu beantworten. Es ergibt keinen Sinn im Kontext eines komponentenbasierten Simulations- und Interaktionskonzeptes, für die computergestützte Biochemie eine weitere Agentenimplementierung der bereits unüberschaubaren Zahl der vorhandenen generischen Agentensysteme hinzuzufügen. Vielmehr ist es Ziel dieses Konzeptes herauszuarbeiten, unter welchen Voraussetzungen *beliebige* Agentensysteme für den Einsatz als Kopplungsagent genutzt werden können.

4.1.2 Phasen einer Multi-Agenten-Simulationsstudie

Die Phasen einer Multi-Agenten-Simulationsstudie in der computergestützten Biochemie unterscheiden sich zunächst wenig von „normalen“ Simulationsstudien. Dies gilt insbesondere für die wichtige Rolle der Modellbildung und Experimentplanung. Auch für Multi-Agentensimulationen gibt es generische Ansätze, und derartig allgemeine Fälle sind oben bereits erwähnt [34].

Erst bei der Betrachtung der tatsächlichen Durchführung des eigentlichen Simulationsexperiments ergeben sich spezifische Vorgehensweisen. Es ist also nötig, ein „Profil“ für die computergestützte Biochemie im Kontext der verschiedenen Phasen einer Multi-Agentensimulation zu finden.

Es lassen sich fünf wesentliche Phasen des Simulationsexperimentes unterscheiden:

1. Startphase
2. Simulationsphase
3. Kommunikationsphase
4. Ergebnisvisualisierung und interaktive Steuerung der Simulation
5. Ende der Simulation.

Die Phasen zwei bis vier wiederholen sich dabei pro Simulationsexperiment vielfach.

In einzelnen Phasen sind folgende Aufgaben zu bearbeiten:

- Startphase: Das Ziel der Startphase ist die Etablierung fester Kommunikationsbeziehungen. Dies bedeutet, dass sich z. B. durch Hierarchisierung und Bildung lokaler Kommunikationscluster Wissen über das Modellsubjekt ausnutzen lässt. Hintergrund ist hier die These dieser Arbeit, dass sich gerade in der computergestützten Biochemie lokale Abhängigkeiten ausnutzen lassen:
    i. Molecular Modeling: Hier findet man lokale Bindungen und Kräfte, die auf kurzen Distanzen wirken.
    ii. Systembiologie: Sowohl Kompartimente als auch örtlich begrenzet und positionierte Simulationsregionen bilden lokale Beziehungen.
    iii. Zellverbände: Ist eine Zelle gleichzeitig eine Simulationsentiät, so lässt sich geometrische Nähe auch in den Kommunikationsstrukturen

abbilden. Auch für die Zelldifferenzierung gilt, dass Tochterzellen ortsnah zu Vaterzellen entstehen, hier können also auch Kommunikationsstrukturen „vererbt“ werden.

Nachdem festgelegt wurde, wie die Kommunikationsstruktur aussieht, **wer** also mit **wem** kommuniziert, muss ausgehandelt werden, **was** während der später folgenden Kommunikationsphasen kommuniziert werden muss.

- Simulationsphase: In der Simulationsphase läuft die eigentliche Simulation ab. Die konkrete Ausgestaltung hängt von verschiedenen Faktoren ab, nicht zuletzt vom im Einzelfall gewählten Simulator oder der Simulationstechnik.
- Kommunikationsphase: Das Ziel der Kommunikationsphase liegt im Austausch der Simulationsrandbedingungen zwischen den Kopplungsagenten. In dieser Phase wird z. B. in Reaktions-Diffusionssystemen die Diffusion von Stoffen kommuniziert.
- Ergebnisvisualisierung und Steuerung: Diese Phase stellt die Verbindung zum Interaktions- und Visualisierungskonzept dieser Arbeit dar und wird im entsprechenden Kapitel und im Kapitel „Integration“ behandelt. Es sei angemerkt, dass bisherige Standardisierungsbemühungen, etwa der FIPA, zu diesem Aspekt keinerlei Aussagen treffen.
- Das Ende der Simulation ist die Phase des Simulationsexperiments, in der Ergebnisse, die über den gesamten Experimentverlauf gesammelt wurden, erhoben und verarbeitet werden. Die konkrete Ausgestaltung ist abhängig vom Einzelfall.

Aus dieser Übersicht wird klar, dass ein wesentlicher Aspekt eines Simulationsexperiments die Kommunikation zwischen den Kopplungsagenten ist. Diese Erkenntnis überrascht nicht, sind doch Agentenkommunikationssprachen (Agent-Communication-Languages, ACL) ein wesentlicher Bestandteil von Multi-Agenten-Systemen. Im folgenden Kapitel wird untersucht, welche Bestandteile der von der FIPA standardisierten ACL im Sinne einer Profilbildung für die computergestützte Biochemie für Multi-Agenten-Simulationsstudien der Biochemie verwendet werden können und welche semantischen Implikationen damit verbunden sind.

4.1.3 Die „BioSLang“ – Bio-Simulation-Language

Im vorigen Kapitel wurde die besondere Rolle der Kommunikation der Kopplungsagenten beschrieben. Die Kommunikation zwischen den Agenten eines Simulationsexperiments der computergestützten Biochemie unterscheidet sich in Art und Inhalt durchaus von der Kommunikation zwischen Agenten in anderem Zusammenhang, z. B. im oben beschriebenen Fall des Trainings in Notfallsituationen.

Das hier vorgestellte Konzept sieht in diesem Fall eine Lösung vor, die die Gemeinsamkeiten sowohl zwischen verschiedenen Fragestellungen eines Teilbereiches, etwa der Systembiologie, als auch zwischen den Bereichen nutzt. Damit scheidet die Spezifikation einer neuen, nur für diesen Fall spezialisierten Agenten-Kommunikationssprache aus, denn dieses Vorgehen entspricht dem oben beschriebenen Denkmuster, dedizierte Insellösungen einem generischen Ansatz vorzuziehen. Der möglicherweise vorhandene Vorteil einer spezialisierten Agenten-Kommunikationssprache eines „schlankeren“, schneller zu implementierenden Systems wird *ad absurdum* geführt, bedenkt man, dass für jeden Fall ein neues System erstellt werden muss, es würde gewissermaßen das Rad jedes Mal neu erfunden.

Eine ähnliche Kritik lässt sich auch für den Fall der Kommunikation zwischen den Kopplungsagenten in der computergestützten Biochemie formulieren: Obwohl der hier betrachtete Anwendungsfall eigene Anforderungen an die Kommunikation stellt, liegt hier im Grunde dennoch „nur“ eine Kommunikation zwischen Agenten vor. Da für den allgemeinen Fall eine Vielzahl von Agentensystemen existiert, die auf einer der standardisierten Agentenkommunikationssprachen operieren, können hier Synergieeffekte und das nicht unerhebliche Fachwissen im Entwurf von Agentenkommunikationssystemen für den Anwendungsfall der computergestützten Biochemie nutzbar gemacht werden.

Dieses Konzept sieht also die Anwendung einer Standard-Agenten-Kommunikationssprache für den Einsatz in der computergestützten Biochemie vor. Damit dies gelingt, ist ein anwendungsspezifisches *Profil* für die Verwendung der Sprache zu finden. Ein *Profil* ist in diesem Zusammenhang ein Vorschlag, wie die Elemente der Standardsprache im Kontext des hier besprochenen Ansatzes genutzt werden können und welche Semantik hierbei verwendet wird.

Wie im Grundlagenkapitel beschrieben, haben sich bisher zwei große Agentenkommunikationssprachen zum *De-facto*-Standard entwickelt: KQML und die von der FIPA standardisierte ACL. Bedenkt man die Ähnlichkeit der beiden Sprachen und die Tatsache, dass die Fortentwicklung der Sprache unter dem Dach der FIPA eine höhere Dynamik zeigt, so scheint es gerechtfertigt, diese Überlegungen anhand der FIPA-ACL durchzuführen.

Das gesuchte Profil besteht aus zwei Teilen:

1. Einsatz und Semantik der Sprachkonstrukte der FIPA-ACL im Kontext der Phasen eines Simulationsexperiments in der computergestützten Biochemie
2. Beschreibung des *Inhalts* (Content) der Kommunikation der Kopplungsagenten

Der erste Teil des Profils besteht aus einer Untermenge der FIPA-ACL Performatives und ihrer Semantik, abgestimmt auf die Bedürfnisse der Simulation in der computergestützten Biochemie. Dieser „Dialekt“ der FIPA-ACL wird im folgenden *Bio-Simulation-Language* oder kurz *BioSLang* genannt.

Die Content-Beschreibung (*Bio-Simulation-Content-Description, BiSCeD*) wird im folgenden Kapitel näher beschrieben. Auf den Inhalt der Kommunikation soll hier vorgegriffen werden, um für die folgenden Ausführungen einen Kontext zu definieren. Informationen über folgende Sachverhalte werden ausgetauscht:

1. Stoffe (Art, Menge etc.)
2. Ortsinformationen (z. B. von Stoffen oder den von der Simulation abgedeckten räumlichen Bereich)
3. Kräfte (z. B. Anziehungskräfte, Felder etc.)

Für jede der oben beschriebenen Phasen des Simulationsexperiments soll nun im Folgenden untersucht werden, welche *Performatives* zu welchem Zweck eingesetzt werden.

- Startphase:
  In dieser Phase „verbreitet“ jeder Agent eine *Liste* von Stoffen und Kräften, die ausgetauscht werden *können*. Diese Informationen stellen den statischen Rahmen des Simulationsexperiments dar. Zusätzlich nennen alle Beteiligten jeweils die Stoffe und Kräfte, die tatsächlich ausgetauscht werden *sollen*. Zusammen und im Zusammenhang mit den Ortsinformationen sind diese Informationen geeignet, Kommunikationsstrukturen zu etablieren oder zu optimieren.

Die Ortsinformationen, die in dieser Phase ebenfalls „verbreitet“ werden, beziehen sich auf den *Ort* und das *Bounding Volume* des von dem jeweiligen Agenten verwalteten Bereichs.
Es sei angemerkt, dass das Zeitintervall zwischen den Phasen in der Regel nicht Gegenstand von Verhandlung ist, sondern anwendungsspezifisch außerhalb dieses Profils gesetzt wird.
In dieser Phase werden die Peformatives

- *Inform* oder *Inform Ref* zur Information über Stoffe, Kräfte und Ort sowie
- *Subscribe* zur Bekanntgabe besonderer Interessen

verwendet.

- Simulationsphase:
  In der Simulationsphase findet keine Kommunikation zwischen den Agenten statt. Die Simulationsphase ist anwendungsspezifisch.
- Kommunikationsphase:
  Die Kommunikationsphase ist die Phase, in der der Hauptteil der Kommunikation stattfindet. Folgende Informationen werden kommuniziert:
  - Information über Kräfte und Felder:
    Genutzte Performatives: *Inform*, *Inform Ref*
  - Information über Stofffluss (keine Ablehnung möglich!)
    Genutzte Performatives: *Inform*, *Inform Ref*
  - Angebot von Stofffluss
    Genutzte Performatives: *Propose*, *Agree*, *Refuse*
  - Nachfrage nach Stofffluss
    Genutzte Performatives: *Propose*, *Agree*, *Refuse*
  - Nachfrage nach Punkt- oder Line-Intersection
    Genutzte Performatives: *Query-If*
  - Information über Ort
    Genutzte Performatives: *Inform*
  - Information über Bounding Volume
    Genutzte Performative: *Inform*
  - Information über neue Agenten:
    Neue Startphase (für diesen Agenten!)

  Der Unterschied zwischen „Information über Stofffluss“ und „Angebot von Stofffluss“ ist sehr wichtig und soll an einem einfachen Beispiel verdeutlicht werden:
  Betrachtet werden soll der Fall der *Diffusion in einem Medium*, also der Fluss einer Substanz, ausgelöst durch *Brown'sche Bewegung* [157] in zwei Konfigurationen. Die erste Konfiguration geht von zwei benachbarten Agenten aus, die jeweils einen Teil des Innenraums einer Zelle beschreiben. Zwischen den Bereichen der Agenten sei keine Barriere, wie etwa eine Membran. In der zweiten Konfiguration sind die gleichen Agenten vorhanden, allerdings wird der Bereich des einen Agenten durch eine Membran begrenzt.
  Im ersten Fall diffundieren Stoffe ungehindert zwischen den beiden Bereichen. Es genügt daher ein *Inform-* oder *Inform-Ref-*Performative.
  Der zweite Fall stellt sich anders dar: Es ist möglich, dass der betrachtete Stoff die

Membran des einen Agenten nicht durchdringen kann. In diesem Fall wird das Angebot des Stoffflusses (*Propose*) abgelehnt werden (*Refuse*). Zu einem späteren Zeitpunkt könnte sich der Zustand der Membran verändert haben, so dass der Stofffluss angenommen wird (*Agree*).
Der wesentliche Unterschied der beiden Fälle liegt in der geringeren Komplexität des Kommunikationsprotokolls im Fall eines einfachen *Inform* (oder *Inform Ref*), das nicht bestätigt werden muss im Gegensatz zu *Propose*, das in jedem Fall beantwortet werden muss.

- Visualisierung und Steuerung
  In der Phase der Visualisierung und Steuerung findet normalerweise keine Kommunikation zwischen den Kopplungsagenten statt. Kommunikation im Zusammenhang der Visualisierung, z. B. das Sammeln der Systemzustände der Agenten, liegt außerhalb dieses Profils.
- Endphase
  Für diese Phase gilt im Wesentlichen das zur Phase „Visualisierung und Steuerung“ Beschriebene.

Aus der Liste der Performatives der FIPA-ACL werden also

- *Susbcribe*
- *Inform, Inform Ref*
- *Propose*
- *Agree*
- *Refuse*
- *Query-If*

verwendet.

4.1.4 Die „BiSCeD“ – Bio-Simulation-Content-Description

Ebenso wichtig wie der Einsatz der Performatives im Kontext der computergestützten Biochemie ist der *Inhalt* (*Content*) der Kommunikation. Die Standardisierungsbemühungen auf der Ebene der Agententechnologie geben hier erwartungsgemäß keine befriedigenden Antworten. Bisher existieren nur wenige Arbeiten zur Vernetzung von Anwendungen im Bereich der computergestützten Biochemie. Den wichtigsten Beitrag liefert hier die Systems Biology Workbench [113], allerdings ist hier nicht an eine Anwendung im Kontext der agentenbasierten Simulation gedacht. Wichtigster Kritikpunkt ist hier jedoch der bewusste Verzicht auf eine formalisierte Inhaltsbeschreibung zu Gunsten eines an Remote Procedure Calls [116] orientierten Ansatzes aus Performanzgründen. Leider bleibt so die Interoperabilität der Anwendungen auf die Kenntnis der Schnittstellen der Anwendungen angewiesen.

Die *Bio-Simulation-Content-Description*, *BiSCeD,* versucht eine formalisierte Beschreibung der im Kapitel „BioSLang“ beschriebenen Kommunikationsinhalte.

Die BiSCeD ist als XML-Dialekt formuliert und lehnt sich in der Namensgebung mancher Attribute an die SBML an, um eine bereits existierende Begriffsfindung zu unterstützen.

Im Folgenden soll die BiSCeD, definiert durch ein XML-Schema [118], genauer beschrieben werden. (Ein unkommentiertes und fortlaufendes Schema sowie die alternative Form einer *Document-Type-Definition (DTD)* findet sich im Anhang.)

Zunächst der *Header* des Schemas:

```
<?xml version="1.0"?>
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema
        targetNamespace="http://www.c-seiler.de/BiSCeD"
        xmlns="http://www.c-seiler.de/BiSCeD">
```

Der Datentyp „positiveFloat" wird später mehrfach verwendet werden:

```
<xs:simpleType name="positiveFloat">
        <xs:restriction base="xs:float">
                <xs:minInclusive value="0.0"/>
        </xs:restriction>
</xs:simpleType>
```

An dieser Stelle beginnt die eigentliche Content-Beschreibung:

```
<xs:element name="BiSCeD">
        <xs:complexType>
                <xs:choice>
                        <xs:element name="Compound" type="CompundType"/>
                        <xs:element name="Space" type ="SpaceType"/>
                        <xs:element name="Field" type ="FieldType"/>
                </xs:choice>
        </xs:complexType>
```

Aus dieser Beschreibung wird die Teilung in drei Bereiche deutlich: Die Kommunikation kann *Stoffe (Compound)*, *Ort und Raum (Space)* sowie *Kräfte und Felder (Field)* beschreiben. Da das <xs:choice>-Konstrukt eine *oder*-Verknüpfung beschreibt, können in einem BiSCeD-Dokument also beliebige Kombinationen dieser komplexen Typen verknüpft werden.

4.1.4.1 Content-Stoffe

Sollen *Stoffe* beschrieben werden, wird der komplexe Typ *CompoundType* verwendet. CompundType besteht aus *ListOfSpecies*-Listen von Stoffen oder aus dem *PutType* oder dem *GetType*. *ListOfSpecies* wird zum Beispiel zur initialen Meldung der möglicherweise austauschbaren Stoffe in der Startphase des Simulationsexperiments eingesetzt. Dagegen werden *GetType* und vor allem *PutType* für den Stoffffluss genutzt (siehe etwa auch das Beispiel der *Diffusion im Medium*, das oben beschrieben wurde).

```
<xs:complexType name="CompoundType">
        <xs:choice>
                <xs:element name="ListOfSpecies" type="ListOfSpeciesType"/>
                <xs:element name="Put" type="PutType"/>
```

```
            <xs:element name="Get" type="GetType"/>
        </xs:choice>
</xs:complexType>
```

Die *ListOfSpecies* enthält, wie der Name bereits verrät, eine Liste von Stoffen. Die Namensgebungen *InitialSpecies* und *InitialAmount* sind an die entsprechenden Begriffe der SBML angelehnt; die Vorsilbe *initial* beschränkt den Einsatz dieses Typs deshalb nicht auf die Startphase des Simulationsexperiments (obwohl er dort am häufigsten Verwendung findet), sondern wurde aus „historischen" Gründen gewählt.

```
<xs:complexType name="ListOfSpeciesType">
        <xs:sequence minOccurs="1" maxOccurs="unbound">
            <xs:element name="InitialSpecies">
                <xs:complexType>
                    <xs:attribute   name="name"
                                    type="xs:string"
                                    use="required"/>
                    <xs:attribute   name="compartment"
                                    type="xs:string"
                                    use="required"/>
                    <xs:attribute   name="initialAmount"
                                    type="positiveFloat"
                                    use="required"/>
                    <xs:attribute   name="boundaryCondition"
                                    type="xs:boolean"
                                    use="required"/>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
</xs:complexType>
```

Besondere Beachtung findet hier der Parameter *boundaryCondition* (wieder der SBML-entlehnt). Ein Wert von *true* stellt hier fest, dass die Menge des in Frage kommenden Stoffs während des Simulationsexperiments *unveränderlich* und damit auch nicht durch Stofffluss austauschbar ist.

PutType und *GetType* sind ganz ähnlich aufgebaut und schließen jeweils Listen von Stoffen ein, die ausgetauscht werden sollen. Der semantische Unterschied zwischen *Get* und *Put* beschreibt die Richtung des Stoffflusses zum die Nachricht sendenden Agenten hin oder von ihm weg.

```
<xs:complexType name="PutType">
```

```
        <xs:sequence minOccurs="1" maxOccurs="1">
                <xs:element name="ListOfVariableSpecies" type=
                "ListOfVariableSpeciesType"/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name="GetType">
        <xs:sequence minOccurs="1" maxOccurs="1">
                <xs:element name="ListOfVariableSpecies" type=
                "ListOfVariableSpeciesType"/>
        </xs:sequence>
</xs:complexType>
```

Die innerhalb eines *Put* beziehungsweise *Get* spezifizierte *ListOfVariableSpecies* beschreibt, welche Stoffe bewegt werden. Die zweite wichtige Information ist dabei die *Menge (Amount)* des umgesetzten Stoffes. Die Stoffmenge wird immer absolut angegeben, also z. B. als Anzahl von Molekülen oder Gewicht. Relative Angaben wie Konzentrationen lassen sich durch die Raum- und Ortsinformationen errechnen.

```
<xs:complexType name="ListOfVariableSpeciesType">
        <xs:sequence minOccurs="1" maxOccurs=„unbound">
                <xs:element name="Species">
                        <xs:complexType>
                                <xs:attribute   name="name"
                                                type="xs:string"
                                                use="required"/>
                                <xs:attribute   name="amount"
                                                type="positiveFloat"
                                                use="required"/>
                        </xs:complexType>
                </xs:element>
        </xs:sequence>
</xs:complexType>
```

4.1.4.2 Content-Raum

Der komplexe Typ *SpaceType* wird genutzt, um Nachrichten zur räumlichen Konfiguration zu codieren. Der Typ kann vier verschiedene Elemente aufnehmen: *SpaceConfiguration*, *Intersection*, *IntersectionLine* und *IntersectionResult*.

Tatsächlich bilden diese Elemente mit den zugehörigen Typen zwei Gruppen: *SpaceConfigurationType* beschreibt die räumliche Position und Ausdehnung von Entitäten, während sich *IntersectionType*, *IntersectionLineType* und *IntersectionResultType* mit der Frage nach „Intersections“ beschäftigen, also mit der Frage, ob ein Punkt in einem räumlichen Gebiet liegt bzw. ob eine Linie ein bestimmtes räumliches Gebiet schneidet.

```
<xs:complexType name=”SpaceType”>
    <xs:choice>
        <xs:element name=”SpaceConfiguration” type=”SpaceConfigurationType”/>
        <xs:element name=”Intersection” type=”IntersectionType”/>
        <xs:element name=”IntersectionLine” type=”IntersectionLineType”/>
        <xs:element name=”IntersectionResult” type=”IntersectionResultType”/>
    </xs:choice>
</xs:complexType>
```

Der *SpaceConfigurationType* enthält eine Liste von Entitäten (*Entity*), für die eine räumliche Konfiguration beschrieben wird.

```
<xs:complexType name=”SpaceConfigurationType”>
    <xs:sequence minOccurs=”1” maxOccurs=„unbound”>
        <xs:element name=”Entitiy” type = “EntityType”/>
    </xs:sequence>
</xs:complexType>
```

Jede Entität wird durch einen Namen eindeutig identifiziert. Darüber hinaus hat jede Entität eine Position (*Point*) und eine Ausdehnung, definiert durch eine Hüllgeometrie, entweder einen Quader (*BoundingBox*) oder eine Kugel (*BoundingSphere)*. Es ist anzumerken, dass die tatsächlich Ausdehnung und Geometrie einer Entität viel komplizierter sein kann (dies liegt in der Verantwortung des zuständigen Agenten); hier wird lediglich eine Hülle definiert.

```
<xs:complexType name=”EntitiyType”>
    <xs:attribute   name=”name”
                    type=”xs:ID”
                    use=”required”/>
    <xs:sequence minOccurs=”1” maxOccurs=”1”>
        <xs:element name=”Point” type=“PointType”/>
        <xs:choice>
```

```
                    <xs:element name=”BoundingBox” type=“BoundingBoxType”/>
                    <xs:element name=”BoundingSphere” type=“BoundingSphereType”/>
                </xs:choice>
        </xs:sequence>
</xs:complexType>
```

Die *attributeGroup* mit dem Namen *xyz* wird als Abkürzung verwendet, wann immer ein Triple von Floating-Point-Werten genutzt werden soll, etwa für die Definition eins Punktes.

```
<xs:attributeGroup name=”xyz”>
        <xs:attribute   name=”x”>
                        type=”xs:float”
                        use=”required”/>
        <xs:attribute   name=”y”>
                        type=”xs:float”
                        use=”required”/>
        <xs:attribute   name=”z”>
                        type=”xs:float”
                        use=”required”/>
</xs:attributeGroup>

<xs:complexType name=”PointType”>
        <xs:attributeGroup ref=“xyz“/>
</xs:complexType>
```

Die Ausdehnung eines Quaders in die drei Raumrichtungen ist durch ein Tripel (x,y,z) beschrieben.

```
<xs:complexType name=”BoundingBoxType”>
        <xs:attributeGroup ref=“xyz“/>
</xs:complexType>
```

Die Hüllkugel wird durch ihren Radius beschrieben. Wie für die *BoundingBox* wird der Mittelpunkt der Geometrie in der Entitätsbeschreibung gesetzt.

```
<xs:complexType name=”BoundingSphereType”>
        <xs:attribute   name=“radius“
                        type=”positiveFloat”
                        use=”required”/>
</xs:complexType>

<xs:complexType name=”IntersectionType”>
        <xs:attribute   name=”id”
                        type=”xs:ID”
                        use=”required”/>
        <xs:attribute   name=”entity”
                        type=”xs:IDREF”
                        use=”required”/>
        <xs:sequence minOccurs=”1” maxOccurs=”1”>
                <xs:element name=”Point” type=”PointType”/>
        </xs:sequence>
</xs:complexType>
```

Die folgenden Typen beschäftigen sich mit Schnitten (*Intersections*), genauer mit der Frage, ob ein gegebener Punkt in einer Entität enthalten ist bzw. ob eine Linie die Entität schneidet. Diese Abfragemöglichkeiten sind nötig, da die räumliche Ausdehnung einer Entität, wie oben beschrieben, in diesem Kontext nur durch eine Hüllgeometrie beschrieben ist, die von der eigentlichen Geometrie abstrahiert. Für grobe Berechnungen genügt diese Hüllgeometrie, für Detailfragen ist ein genaueres Werkzeug nötig.

Die *Intersection*-Typen beschreiben die Entität, gegen die der Test ausgeführt werden soll, eine ID, um eine spätere Antwort zuordnen zu können, und, je nach Ausprägung, einen oder zwei Punkte.

```
<xs:complexType name=”IntersectionLineType”>
        <xs:attribute   name=”id”
```

```
                    type=”xs:ID”
                    use=”required”/>
        <xs:attribute   name=”entity”
                    type=”xs:IDREF”
                    use=”required”/>
        <xs:sequence minOccurs=”2” maxOccurs=”2”>
                <xs:element name=”Point” type=”PointType”/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name=”IntersectionResultType”>
        <xs:attribute   name=”intersection”
                    type=”xs:IDREF”
                    use=”required”/>
        <xs:attribute   name=”value”
                    type=”xs:boolean”
                    use=”required”/>
</xs:complexType>
```

4.1.4.3 Content-Kräfte und Felder

Die Beschreibung zu Kräften und Feldern, zusammengefasst durch den *FieldType*, kennt zwei Möglichkeiten der Spezifikation einer Kraft bzw. eines Feldes: *Singletons* und *Arrays*.

Für alle Felder gilt: Die Arten, sprich die Namen eines Feldes, und seine physikalische(n) Einheit(en) sind nicht normiert, sondern werden in einem Attribut namens *Type* definiert.

```
<xs:complexType name=”FieldType”>
        <xs:choice>
                <xs:element name=”Singleton” type=”SingletonType”/>
                <xs:element name=“array” type=”ArrayType”/>
        </xs:choice>
</xs:complexType>
```

Ein *Singleton* definiert sich durch einen Punkt und einen Wert. Die Wirkungen und Reichweite eines solchen Feldes sind abhängig von seinem Typ und werden von der BiSCeD nicht erfasst.

```
<xs:complexType name=”SingletonType”>
```

```
        <xs:attribute    name=”type”
                         type=”xs:NMTOKEN”
                         use=”required”/>
        <xs:attribute    name=”value”
                         type=”xs:double”
                         use=”required”/>
        <xs:sequence minOccurs=”1” maxOccurs=”1”>
                <xs:element name=”Point” type=”PointType”/>
        </xs:sequence>
</xs:complexType>
```

Der *Array*-Feldtyp wird durch eine Liste von *Elements* spezifiziert. Jedes *Element* ähnelt dabei einem Singleton. Zusätzlich zu Ort (*Point*) und Wert wird hier jedoch noch ein Vektor angegeben, der die Richtung des Feldes am angegebenen Punkt beschreibt.

```
<xs:complexType name=“array”>
        <xs:sequence minOccurs=”1” maxOccurs=„unbound”>
                <xs:element name=”Element“ type=“ElementType“/>
        </xs:sequence>
<xs:complexType>


<xs:complexType name=ElementType>
        <xs:attribute    name=”type”
                         type=”xs:NMTOKEN”
                         use=”required”/>
        <xs:attribute    name=”value”
                         type=”xs:double”
                         use=”required”/>
        <xs:sequence minOccurs=”1” maxOccurs=”1”>
                <xs:element name=”Vector” type=”VectorType”/>
                <xs:element name=”Point” type=”PointType”/>
        </xs:sequence>
</xs:complexType>

<xs:complexType name=VectorType>
        <xs:attributeGroup       ref=”xyz”/>
</xs:complexType>
```

Das Tag-</xs:schema> beendet das XML-Schema der BiSCeD.

4.1.5 Anmerkung zur Kopplung zeitdiskreter und zeitkontinuierlicher Simulationen

Das hier vorgestellte Simulationskonzept macht keine Annahme bezüglich der Art der verwendeten Simulationstechnik der verschiedenen Entitäten. Tatsächlich ist das Konzept dazu gedacht, verschiedenartige Simulationstechniken miteinander zu verbinden. Auch in anderen Arbeiten [111], [111a] ist die Frage nach der Kombination verschiedene Techniken diskutiert worden, der Ansatz ist dort jedoch vollständig verschieden im Vergleich mit dem hier vertretenen Ansatz der gekoppelten Agenten. Dort wird versucht, eine rein mathematische Überführung von zeitdiskreten Simulationen in zeitkontinuierliche Simulationen zu implementieren. Die vorgestellten Lösungen sind in der technischen Realisierung durch die nötige Zwischenschaltung eines Integrators aufwendiger als das in dieser Arbeit vorgestellte Konzept.

Diese Arbeit geht den entgegengesetzten Weg und macht dabei von einer pragmatischen Beobachtung Gebrauch. Auch zeitkontinuierliche Modelle werden in der numerischen Berechnung nur für diskrete Zeitpunkte gerechnet. Dieses Verfahren gleicht einer Abtastung der kontinuierlichen Funktion oder Funktionen und damit einer Diskretisierung.

Betrachtet man nun zeitdiskrete Systeme und tastet den Systemzustand auf die gleiche Art wie eben beschrieben ab, so lässt sich allein anhand der abgetasteten Werte nicht mehr erschließen, auf welche Art die Werte berechnet wurden.

Im Sinne eines komponenten- und agentenbasierten Konzeptes, das den inneren Zustand der Komponente bzw. des Agenten nach „außen“ kapselt, ist die Verbindung verschiedener Simulationstechniken transparent (siehe auch Abbildung 64).

Die Wahl des Kommunikationsintervalls ist von großer Bedeutung. Wie weiter oben beschrieben wurde (Kapitel 4.1.3), ist diese Wahl aber vom konkreten Simulationsexperiment abhängig. Weiterhin können die konkret gewählten Simulationsverfahren Anforderungen an die Kommunikationsfrequenz stellen, die bei der Kopplung entsprechend aufeinander abzustimmen sind.

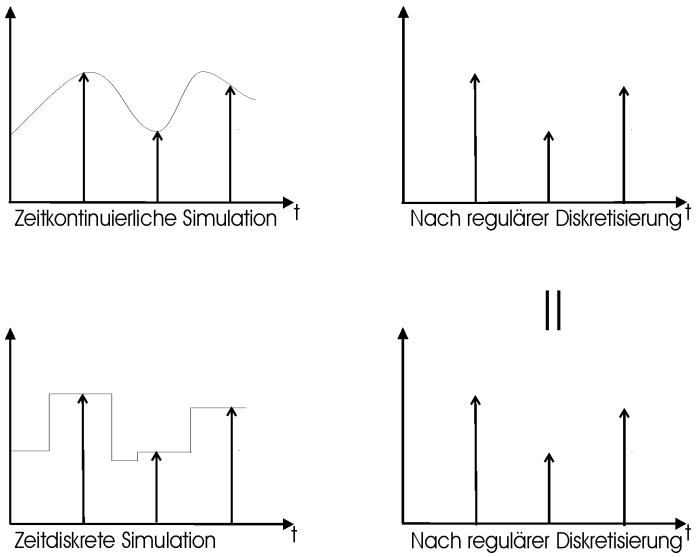


Abbildung 64 Übergang zeitdiskret abgetastet -
zeitkontinuierlich abgetastet

4.2. Konzept – Interaktion und Visualisierung

Analog zur Simulation ist für die Interaktion und Visualisierung ein Konzept zu erstellen, das die Anforderungen berücksichtigt, die sich aus dem Anwendungsgebiet der computergestützten Biochemie ergeben. Der bereits zu Beginn dieses Hauptkapitels eingeführte Ansatz mittels Komponenten und Agenten wird hier fortgeführt.

Insbesondere werden hier zunächst die Aspekte behandelt, die ein derartiger Ansatz für die Visualisierung bedeutet. Danach wird ein Interaktionskonzept dargestellt, das ein neues Ein-/ Ausgabegerät, die „Virtual Glove Box“, beinhaltet, das hier vorgestellt wird.

4.2.1 Visualisierung

In den vorangegangenen Kapiteln wurde der Begriff der *Entität* bereits mehrfach eingeführt. Es wurden für die verschiedenen Anwendungsbereiche im Anwendungsfeld der computergestützten Biochemie verschiedene *Entitäten* identifiziert und umrissen. *Entitäten* können grob als *Elemente einer Visualisierung mit abgrenzbaren Verhalten* definiert werden.

Hier soll nun der Begriff der Entität für die Visualisierung genauer untersucht werden. Dabei wird der Rahmen der computergestützten Biochemie verlassen und eine allgemeine Klassifikation durchgeführt werden.

Im Folgenden soll untersucht werden, wie Entitäten klassifiziert werden können und welche Maßnahmen für die Kombination der Entitäten zu einer Visualisierungsanwendung notwendig sind.

Eine Applikation enthält im Wesentlichen zwei verschiedene Gruppen von Entitäten, deren Verhalten sich beschreiben lässt:

- Entitäten mit visueller Repräsentation
- Entitäten ohne visuelle Repräsentation.

Visuelle Repräsentation bedeutet hier, dass die Entitäten in der Visualisierung durch visuelle, akustische, haptische und weitere Modelle repräsentiert werden und somit direkt vom Benutzer wahrgenommen werden können. Innerhalb der beiden Gruppen lassen sich wiederum verschiedene Untergruppen identifizieren.

Eine erste Unterscheidung lässt sich für Objekte mit einer visuellen Repräsentation über die Dynamik eines Objektes treffen. So gibt es in den meisten Anwendungen Teile der Szene, die über den gesamten Verlauf der Anwendung unveränderlich, also statisch sind, während andere ein dynamisches Verhalten zeigen. Zwar ist diese Unterscheidung unter konzeptionellen Gesichtspunkten weniger bedeutend, schließlich ist auch „statisch“ eine Verhaltensbeschreibung; allerdings ist die Unterscheidung in Bezug auf die Optimierung der Performanz der Anwendung durchaus relevant.

Unter den dynamischen Entitäten lassen sich solche unterscheiden, die direkt durch Eingaben des Benutzers gesteuert werden, und solche, deren Verhalten nur mittelbar vom Benutzer

abhängt. Die zweite Kategorie reicht von einfachen „unbelebten“ Gegenständen, z. B. Türen, Schaltern etc., bis zu „Mitspielern“, wie etwa Menschen, Tieren etc.

Während in vielen Visualisierungsapplikationen der Benutzer selbst keine Repräsentation in der Szene hat und frei durch die Szene navigieren kann, zeichnen sich z. B. Spielanwendungen meist dadurch aus, dass der Benutzer in der Szene repräsentiert wird. Gleiches gilt für interaktive Anwendungen, in denen der Benutzer verändernd in die Szene eingreift. Für diese Repräsentation ist der Begriff des Avatars geprägt worden. Der Avatar des Benutzers wird direkt von den Eingaben des Benutzers gesteuert. Er ist jedoch in der Regel nicht frei in der Navigation, sondern wird durch die der Anwendung zugrunde liegende Konzeption eingeschränkt. Diese Einschränkung lässt sich ebenfalls durch Verhaltensbeschreibungen definieren, die auch in einem Agenten gekapselt sein können.

Die Objekte, die keine Repräsentation in der Szene haben, reichen von Diensten wie einer Zeitrepräsentation über Felder (elektromagnetische Felder, Gravitation, Druck etc.) bis zum Zugang zu externen Dienstleistungen, wie Berechnungen oder Datenbankabfragen. Diese Gruppe von Entitäten ist zu heterogen, um weitere Unterscheidungen zu treffen. Eine Ausnahme hiervon bilden alle Dienste, die im Zusammenhang mit Ein- oder Ausgabeverhalten stehen. Diese Gruppe wird wegen ihrer besonderen Bedeutung im nächsten Kapitel gesondert betrachtet. Abbildung 65 zeigt die Klassifikation noch einmal schematisch.

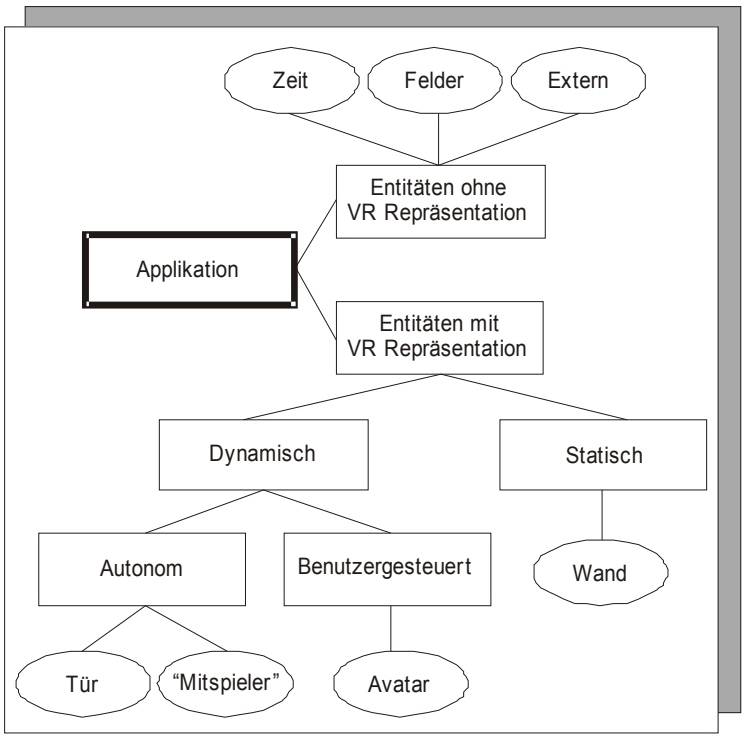


Abbildung 65 Bild Klassifikation Entitäten

Sollen die verschiedenen Entitäten in einer Applikation kombiniert werden, so sind Maßnahmen zu treffen, um ein Zusammenwirken zu ermöglichen. Zusammenwirken bedeutet, dass sich die Entitäten gegenseitig beeinflussen können; hierzu müssen sie Informationen

untereinander austauschen können. Kommunikation ist also das wichtigste Bindeglied zwischen den Entitäten.

Betrachtet man die verschiedenen Kategorien von Entitäten, so fällt auf, dass es verschiedene Kommunikationsbedürfnisse unter ihnen gibt. Verfolgt man den beschriebenen Ansatz der intelligenten Agenten weiter, so wird jede Entität, wie beschrieben, durch einen Agenten gekapselt. Durch diese Kapselung gibt es zwischen den Entitäten eine einheitliche Kommunikationsschnittstelle.

Die Kommunikation zwischen den Entitäten erfolgt nun durch Austausch von Nachrichten untereinander. Für die Definition der Schnittstelle ist es dabei gleichgültig, ob die gekapselte Entität eine Repräsentation hat oder nicht oder ob ihr Verhalten durch ein Regelwerk, durch Simulation oder direkt über die Eingaben des Benutzers gestaltet wird. Diese Kommunikation bezeichnet man als abstrakt-logische Kommunikation. Sie findet auf Entitätsebene statt und hat Inhalte wie beispielsweise: „Stoff X diffundiert von A nach B“. Genaueres zu der Art der Kommunikation wurde bereits in den Kapiteln zur „BioSLang“ und der „BiSCeD“ beschrieben.

Als Kommunikationsstruktur wird im allgemeinen Fall eine N-zu-M-Struktur, wie etwa ein Kommunikationsbus, vorgeschlagen. Über die Möglichkeiten der Optimierung der Kommunikationsstruktur wurde bereits bei der Besprechung der Phasen eines Simulationsexperiments in den vorhergehenden Kapiteln berichtet.

Trotz einfacher Schnittstelle in Form von Weitergabe von Nachrichten in einem einheitlichen Format bleibt das Problem der Interpretation ausgetauschter Nachrichten. Dieses Problem ist identisch mit dem Problem der Kommunikation von Menschen in natürlicher Sprache. Zwar ist auch hier die Schnittstelle hinreichend definiert, allerdings hängt der Kommunikationserfolg nicht zuletzt von der Benutzung eines gemeinsamen Wortschatzes und Kommunikationsformates ab. Für den abgrenzbaren Fall der computergestützten Biochemie wurde dieser Fall in der oben beschriebenen Konzeption behandelt. Eine allgemeine Lösung ist hier nicht möglich.

Die Struktur einer abstrakt-logischen Kommunikation ist nicht für jede notwendige Kommunikation zwischen Entitäten geeignet. Dies gilt insbesondere für die Kommunikation im Rahmen der Ein- und Ausgabe, für die daher gegebenenfalls eigene Kommunikationsstrukturen zu erstellen sind.

4.2.1.1 Verknüpfung der Entitäten mit der Visualisierungsinfrastruktur

Im vorigen Kapitel wurde mehrfach auf die besondere Rolle und die besonderen Bedürfnisse der Ein- und Ausgabe hingewiesen. Diese Rolle ist deswegen eine Ausnahme und bedarf besonderer Betrachtung, weil die Ein- und Ausgabe der Ausgangspunkt der gesamten betrachteten Visualisierungsanwendung ist. Ein Fehlen oder ein Mangel in diesem Bereich stellt die gesamte Applikation in Frage, selbst wenn alle anderen Aspekte hinreichend bearbeitet wurden.

Ein wichtiger Aspekt ist die Frage der Kommunikationsstruktur. Eine Nutzung des oben beschriebenen logischen Kommunikationsbusses ist nicht geeignet. Das notwendige Kommunikationsvolumen ist in der Regel sehr viel höher als das restliche Kommunikationsvolumen. Für die visualisierungsspezifische Kommunikation wird daher eine eigene Kommunikationsstruktur vorgeschlagen, wie sie bereits oben für Dienste mit besonderen Kommunikationsanforderungen eingeführt wurde [4].

Alle bisher betrachteten Anforderungen der Ein- und Ausgabe lassen die Integration in das Gesamtframework offen. Für die Einordnung der Ein- und Ausgabe lassen sich zwei wesentliche Alternativen erkennen. Zum einen kann die E/A wie eine normale Szenenentität ohne visuelle Repräsentation gesehen werden (keine eigene Repräsentation, da die Aufgabe der E/A ja gerade in der Repräsentation der *anderen* Entitäten liegt), zum anderen lässt sich die E/A als eigene Kategorie definieren (siehe Abbildung 66).

Betrachtet man die Ein- und Ausgabe als Dienst wie jeden anderen in der Applikation, so ist damit eine Reihe von Vor- und Nachteilen verbunden. Die Vorteile dieser Einordnung liegen darin, dass die beschriebene Klassifikation und das Konzept beibehalten werden. Die Applikation wird über ihre logische Struktur und ihre Entitäten definiert. Soll die Applikation für eine andere VR-Umgebung portiert werden, so ist dies ohne Brüche möglich, denn hierzu ist lediglich der E/A-Baustein zu ersetzen.

Der Nachteil dieses Vorgehens liegt darin, dass das Agentenkonzept in einem wesentlichen Punkt durchbrochen wird: Der E/A-Dienst interpretiert die logischen Nachrichten der anderen Entitäten („ich, Entität A, habe dieses Aussehen“). Die Entitäten sind also nicht mehr autonom in der Hinsicht, dass sie auf ihre tatsächliche Repräsentation keinen direkten Einfluss mehr haben. Darüber hinaus muss die E/A-Entität Zusatzinformationen haben, wie die logische Nachricht zu interpretieren ist, denn die Bedeutung der Nachrichten ist, wie oben genannt, kontextabhängig.

Um wiederum entsprechende Hilfestellung geben zu können, muss der Agent genügend Wissen um die Möglichkeiten der Visualisierungsschnittstelle haben. Dies stellt letztendlich doch einen Bruch des Konzepts dar, das hier eigentlich eine Abstraktion von der konkreten E/A-Konfiguration fordert.

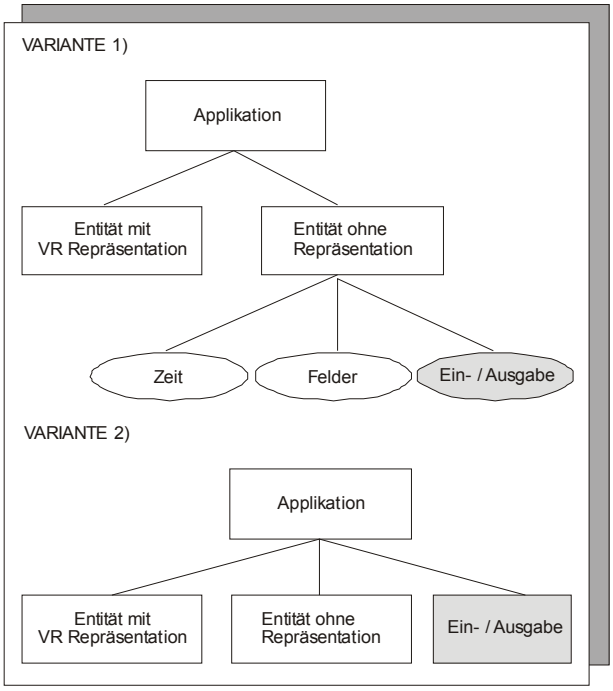


Abbildung 66 Einordnung der Ein-/ Ausgabe in die Klassifikation

Hebt man dagegen die E/A-Komponente als besonderen Dienst hervor, d. h., gibt man den Agenten besonderes Wissen über die eingesetzte E/A-Schnittstelle mit, so verändert sich das Konzept wie folgt: Jeder Agent hat einen E/A-Aspekt zusätzlich zu der oben beschriebenen logischen Kommunikationsschnittstelle. Der Agent bestimmt über seine Repräsentation selbst, indem er die E/A-Schnittstelle direkt ansteuert. Zusätzlich überträgt er Nachrichten über den logischen Kommunikationskanal. Der Vorteil eines solchen Konzepts liegt in einer potentiell besseren Performanz der Applikation, da der kritische Aspekt der Ein- und Ausgabe direkt gesteuert wird. Zudem wird nun auch das Autonomiegebot aus dem Agentenkonzept beachtet. Nachteil dieser Vorgehensweise ist, dass die einzelnen Entitäten nun an die verwendete E/A-Schnittstelle gebunden sind. Die Wiederverwendbarkeit der Bausteine wird somit eingeschränkt. Eine Veränderung der E/A-Konfiguration hat in der Regel eine Veränderung der einzelnen Agenten zur Folge. Die Komplexität der Agenten erhöht sich.

Beide Verfahren haben ihre Berechtigung; es ist nicht möglich, eine allgemeine Bewertung zu treffen. Die Auswahl des konkreten Verfahrens muss vor dem Hintergrund der Anwendung getroffen werden. Während die erste Variante robuster gegenüber Veränderungen ist und eine schnellere Erweiterung zulässt, zeigt die zweite Variante eine bessere Performanz und setzt das agentenbasierte Grundkonzept besser um.

4.2.2 Interaktion

Neben der konzeptionellen Betrachtung der Visualisierung ist auch ein Interaktionskonzept notwendig. Hierzu muss betrachtet werden, wie ein Nutzer in der computergestützten Biochemie mit einer Simulationsapplikation, wie sie in dieser Arbeit beschrieben wurde, interagiert.

Es wurde in den vorhergehenden Kapiteln mehrfach auf den dreidimensionalen Charakter des Anwendungsgebietes eingegangen und kritisiert, dass Visualisierung und Simulation hierauf nicht entsprechend Rücksicht nehmen. Auch eine möglichst intuitive Interaktion muss auf diese Eigenschaft eingehen. Betrachtet man nun aber die traditionellen Ein- und Ausgabegeräte und die zugehörigen Metaphern, wie etwa Bildschirm, Tastatur und Maus mit der „Desktop“-Metapher, so zeigt sich schnell, dass diese nur bedingt für den Einsatz in 3D geeignet sind [158].

Abbildung 67 Spacemouse [159] (links) und Phantom [160] (rechts)

Selbst 3-D-Eingabegeräte, wie etwa die Spacemouse [159] oder PhANToM [160] (Selbst 3-D-Eingabegeräte, wie etwa die Spacemouse [159] oder PhANToM [160] (), sind für den Anwendungsfall nicht sehr gut geeignet. Es zeigt sich, dass die natürliche Interaktionsform im betrachteten Bereich durch Greifen charakterisiert werden kann, also z. B. „Halten eines Moleküls, während ein Atom bewegt wird“ oder „zwei Zellen voneinander lösen“. Die

genannten Geräte erlauben eine solche im wörtlichen Sinne direkte „Manipulation“ nicht.), sind für den Anwendungsfall nicht sehr gut geeignet. Es zeigt sich, dass die natürliche Interaktionsform im betrachteten Bereich durch *Greifen* charakterisiert werden kann, also z. B. „Halten eines Moleküls, während ein Atom bewegt wird“ oder „zwei Zellen voneinander lösen“. Die genannten Geräte erlauben eine solche im wörtlichen Sinne direkte „Manipulation“ nicht.

Die genannten Geräte beschreiben die Möglichkeiten auf der Eingabeseite. Für die Ausgabeseite steht in der Regel „nur“ ein 2-D-Anzeigegerät, wie etwa ein Bildschirm, zur Verfügung.

Im Rahmen dieser Arbeit wird daher eine neue Metapher für die Ein- und Ausgabe vorgeschlagen: die *Virtual Glove Box*. Dieses Konzept besteht aus drei Teilen:

1. einem Konzept für die Eingabegeräte,
2. einem Konzept für die Ausgabegeräte,
3. einer gemeinsamen Metapher, die die Geräte logisch in den Anwendungskontext einbettet.

Das Konzept für die Eingabegeräte beinhaltet die Forderung direkter, manipulativer Eingriffsmöglichkeit des Benutzers. Die Umsetzung dieser Forderung wird unten und im Kapitel zur Realisierung genauer beschrieben; dort wird auch ein passendes passives 3-D-Stereosystem für die 3-D-Ausgabe präsentiert.

Die hier vorgeschlagene Metapher ist die einer Glove Box. Das Prinzip einer solchen Glove Box ist es, den Benutzer vom Arbeitsbereich zu trennen und ihm dennoch die Möglichkeit zu geben, die zu bearbeitenden Substanzen direkt zu manipulieren. Eine solche Trennung von Benutzer und Arbeitsbereich ist immer dann nötig, wenn eine direkte Handhabung der Substanzen und Gegenstände für den Benutzer zu gefährlich ist (Substanzen sind giftig, es treten Gase aus etc.) oder wenn das Arbeitsmaterial vom Benutzer oder der Umgebung kontaminiert werden würde (Arbeit in geschlossener oder abgeschlossener Atmosphäre). Der Aufbau eines solchen Apparates besteht aus einer durchsichtigen Kammer, in deren Seitenwand Öffnungen vorgesehen sind. Diese Öffnungen dienen entweder der Beladung (sie sind für die Umsetzung als VR-System nicht relevant) oder sind Einlässe in Handschuhe. Durch diese Handschuhe greift der Benutzer in die Kammer, ohne mit dem Inhalt direkt in Kontakt zu kommen (siehe Abbildung 68).

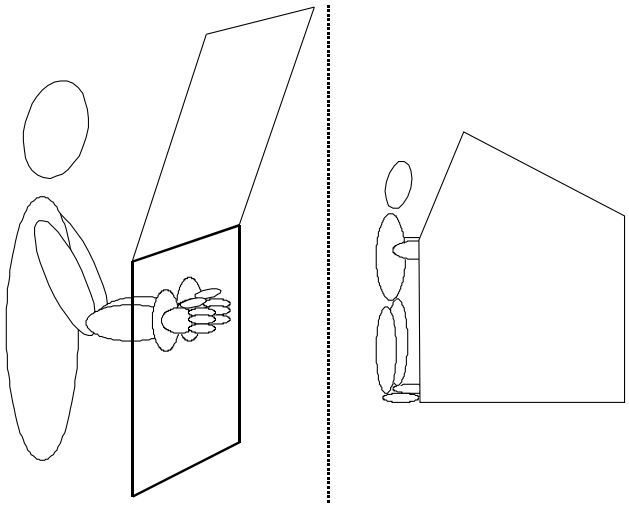


Abbildung 68 Schema (Virtual) Glove Box

Diese Metapher ist aus dem Anwendungsbereich der Biochemie entnommen und erleichtert somit dem fachlich versierten Benutzer die Arbeit. Diese Metapher hat auch für die Umsetzung in computergraphische Technologie Vorteile; so sind durch die fixe Montage der Handschuhe die Freiräume für die Bewegung von Händen und Armen hinreichend eingeschränkt und für den Benutzer einer Glove Box auch üblich. Dies erleichtert das Registrieren und Verfolgen der Bewegungen des Benutzers.

Die zentrale Forderung der „hands on"-Interaktion wird aus Sicht der technischen Umsetzung in zwei Anforderungen übersetzt: Zum einen müssen Position, Haltung und Bewegung der Hände und der Finger des Benutzers ständig erfasst werden, zum anderen muss es eine direkte Rückkopplung zum Benutzer geben, wenn dieser ein virtuelles Objekt berührt.

In diesem Anwendungsfall wird der Vorteil der haptischen Darstellung deutlich, wie sie im Grundlagenkapitel eingeführt wurde: Ohne Umwege über wenig intuitive Eingabegeräte, wie Maus oder Spacemouse, ist ein direktes „Begreifen" und „Manipulieren" der Entitäten möglich.

Aus dem vorher Gesagten ergibt sich die Aufgabe des konzeptionellen Aufbaus einer Benutzungsschnittstelle für spezifische Anwendungen der Virtual Glove Box. Da dem Benutzer virtuelle Hände als Eingabegeräte zur Verfügung stehen, muss demzufolge jedes Schnittstellenelement mit diesen bedient werden können. Ein Blick zurück auf die Grundlagen zeigt zwei grundsätzliche Methoden zur Anwendungssteuerung:

- Benutzen der Hände zum Steuern von Eingabegeräten, welche mit Hilfe spezieller Interaktionsmetaphern die Steuerung der Anwendung ermöglichen
- Benutzen virtueller Hände, welche durch Einsatz von Gesten die Steuerung einer Anwendung ermöglichen.

Der hier vorgestellte Ansatz soll die Vorteile beider Methoden vereinen: die Universalität der Eingabegeräte und das Einbinden des Benutzers in die Szene mit Hilfe virtueller Hände.

Durch den Einsatz der virtuellen Hände ergibt sich für die Bedienung eine Kombination von direkter Manipulation und dem Einsatz virtueller Werkzeuge, welche wiederum direkt durch die Hände manipuliert werden können. Der Benutzer soll das jeweils passende Werkzeug für die anfallenden Aufgaben zur Verfügung gestellt bekommen.

Durch diesen Ansatz, welcher nur durch den Einsatz von Force-Feedback möglich ist, wird die Anwendungssteuerung nicht auf hochspezialisierte Aufgaben beschränkt. Es ist sowohl die natürliche Interaktion mit Objekten möglich (wie in der realen Welt) als auch die Ausführung abstrakter Aufgaben, wie das Einschalten einer Lampe über einen virtuellen Schalter oder auch der Einsatz eines virtuellen Hammers zur Zerstörung eines Objekts. Auch die Simulation einer herkömmlichen Maus ist denkbar. Ebenso ist die zweihändige Bedienung von Werkzeugen möglich. Die Werkzeuge, welche einem realen Pendant entsprechen können oder speziell für eine Aufgabe entworfen sind, können zusätzlich mit in der realen Welt unmöglichen Fähigkeiten ausgestattet sein. So kann ein Hammer über das Wissen verfügen, dass er keine Objekte aus Glas zerstören darf. Er kann sich auch seinen Aufbewahrungsort merken und sich nach dem Loslassen dorthin begeben.

4.2.2.1 Komponenten und Frameworks

Diese Arbeit orientiert sich auch an den Arbeiten [1], [2], deren Konzepte über 3-D-Beans und 3-D-Komponenten einen Schwerpunkt darauf gelegt haben, dass die entwickelten 3-D-Komponenten bereits während des Erstellungsprozesses genutzt werden können, so dass die Erstellung bereits im virtuellen Raum erfolgen kann. Dabei kapseln 3-D-Komponenten Verhalten und Geometrie und stellen die dazu benutzten Schnittstellen sowohl für den Benutzer wie auch für das 3-D-Framework zur Verfügung. Dieser äußere Rahmen stellt die Infrastruktur für die 3-D-Komponenten zur Verfügung. So kann beispielsweise über das 3-D-Framework auf das zugrunde liegende 3-D-Graphiksystem oder auf die Kommunikation zugegriffen werden. Für die hier vorgestellte Arbeit war es notwendig, diese Konzepte in der Weise zu erweitern, dass mit dem Einfügen einer Komponente nicht nur ein visuelles Objekt mit dem dazugehörigen Interaktions- und Simulationsverhalten eingefügt wird, sondern dass sowohl der Tracker als auch die Ansteuerung in die haptischen Ausgabegeräte (Exoskelette) integriert wird. Eine Komponente im Sinne dieser Arbeit ist somit ein Software-Modul, das von der visuellen Repräsentation als Frontend über die Verhaltensbeschreibung als Programmlogik bis hin zur Ansteuerung der Graphik und der Haptik als Backend reicht.

4.2.2.2 EASY: Ein “greifbares” graphisches User-Interface

Aufgrund der vorangegangenen Überlegungen entstand ein Konzept, welches den dargelegten Möglichkeiten gerecht werden soll. Die Benutzungsoberfläche bietet die Möglichkeit zur zweihändigen und direkten Interaktion. Die Entwicklung der Datenstrukturen ist von den Vorgaben der benutzten Bibliotheken gekennzeichnet.

Die Entwicklung wiederverwendbarer Oberflächenelemente impliziert die Erstellung einer Bibliothek zur Speicherung aller notwendigen Komponenten. So kann der Entwickler einer Anwendung die Komponenten und die nötige virtuelle Umgebung ohne Quellcode-Kenntnisse leicht in sein Projekt einfügen. Die statische Konfiguration der Virtual Glove Box (Nutzung stets gleicher Hardware) spricht für die Bereitstellung eines vorgefertigten Simulationsablaufs, so dass sich der Entwickler ausschließlich um das Verhalten der Komponenten seiner Simulation zu kümmern braucht.

Benötigt wird also eine Grundkomponente, welche der Entwickler einer Simulation nach seinen Bedürfnissen erweitern und anpassen kann. Sie sollte Wissen über ihre visuelle und haptische Repräsentation kapseln, so dass die Aktualisierung der beiden Szenegraphen automatisch erfolgen kann oder vom Entwickler nur angestoßen werden muss. Im Idealfall sollten sich aus ihr alle Bedien- und Simulationselemente ableiten lassen. Des Weiteren sollte sie über eine Methode verfügen, welche bei jedem Durchlauf der Simulation aufgerufen wird und die Ausführung beliebiger Methoden ermöglicht.

Für die Steuerung von Anwendungen ist in irgendeiner Form immer der Einsatz von Schaltern oder Tasten notwendig. Auch Schiebe- oder Drehregler finden sich an vielen Geräten. Da sich auch in der realen Welt keine Alternativen finden, ist demnach die Implementation solcher Elemente erforderlich.

Die Umsetzung des Konzeptes ist im Kapitel zur Realisierung genauer beschrieben.

4.3. Integration der Konzepte

In den vorherigen Kapiteln wurde jeweils ein Konzept für die Simulation sowie ein Konzept für die Visualisierung und die Interaktion vorgestellt. Wie bereits in der Einleitung zu diesem Kapitel vermerkt wurde, laufen diese Konzepte auf ein gemeinsames Konzept zusammen.

Die Entitäten der Visualisierung verschmelzen mit den Agenten der Simulation und den Komponenten der haptischen Darstellung zu einem gemeinsamen Ansatz. Es entsteht ein Kopplungsagent, der drei wesentliche Aufgaben übernimmt:

1. Kopplung der Simulationen, d. h. Aufbau und Teilnahme an einer Kommunikationsstruktur, über die simulationsspezifische Informationen ausgetauscht werden
2. visuelle Darstellung der Simulationsentität im Zusammenhang eines Visualisierungsrahmens
3. haptische Darstellung der Simulationsentität in einem dedizierten Haptik-Framework.

Abbildung 69 verdeutlicht den Zusammenhang, der im Folgenden genauer betrachtet wird.

Zur Beschreibung des Aufbaus eines Kopplungsagenten ist daher eine differenzierte Betrachtung seines *Verhaltens* notwendig. Es werden im Rahmen der Umsetzung der Glove-Box-Metapher die folgenden Aufgaben unterschieden:

- visuelles Verhalten
- Haptikverhalten
- Interaktionsverhalten
- physikalisches, biologisches und chemisches Verhalten
- Kommunikationsverhalten.

Die Visualisierung wird nicht von einer zentralen Einheit übernommen, sondern jede Komponente stellt sich selbst dar, indem sie eine definierte Schnittstelle für das zugrunde liegende 3-D-Graphiksystem bietet. Die Grundlage für das dazu notwendige visuelle Verhalten bilden geeignete Methoden zur Visualisierung. Im Beispiel des Molecular Modelings stehen hierzu Methoden wie „Ball and Stick“, „Ball“ oder „Wire Frame“ zur Verfügung [6]. Das Haptikverhalten beschreibt die Reaktionen einer Komponente auf die getrackten Bewegungen eines Benutzers ebenso wie die Reaktionen der haptischen Ausgabegeräte. Es stellt somit eine besondere Form des Interaktionsverhaltens dar, das die Reaktion des Systems auf die Benutzeraktionen beschreibt. Die physikalisch, biologisch und chemisch basierten Simulationen beschreiben das Systemverhalten und greifen dabei wiederum auf entsprechende

Simulatoren zurück. Das Kommunikationsverhalten beschreibt das Zusammenspiel mit anderen Komponenten und mit dem darunterliegenden Framework.

Dieser Ansatz des Agenten, der wiederum in sich aus drei Unterkomponenten besteht, lässt Raum für den Aufbau eines Baukastens, in dem sich verschiedene Simulatoren mit verschiedenen Visualisierungen und haptischen Repräsentationen verbinden lassen.

Das Gesamtkonzept wäre nur sehr eingeschränkt nutzbar, wenn es ein multimodales Display und damit haptische Ein- und Ausgabegeräte zwingend vorschreiben würde, denn derartige Displays sind teuer, aufwändig und nur in geringer Anzahl verfügbar. Hier hilft allerdings der Unterkomponentenansatz: Ein Kopplungsagent muss adaptiv die zu Verfügung stehende Infrastruktur berücksichtigen.

Praktisch bedeutet dies, dass der Einsatz der Haptik optional ist (siehe Abbildung 70). Man könnte argumentieren, dass auch die Visualisierungskomponente nicht notwendigerweise Teil jeder Applikation sein muss. Würde man allerdings auch die Visualisierungskomponente als optional einstufen, verlöre man gegebenenfalls all jene Vorteile der Visualisierung, die eine erfolgreiche Simulationsstudie unter Berücksichtigung räumlicher Eigenschaften des Modellsubjekts erst möglich machen. Wie in den Kapiteln zur Anforderungsanalyse berichtet, würde man damit auch die Simulation beschneiden.

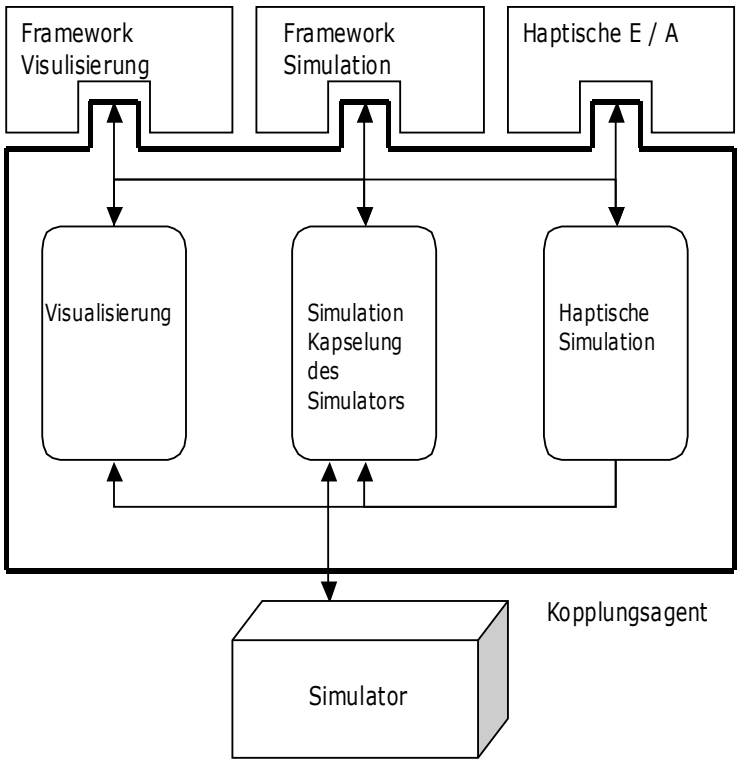


Abbildung 69 Kopplungsagent mit Haptik

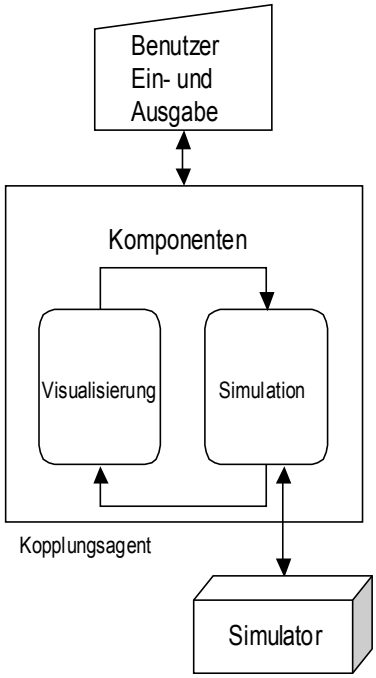


Abbildung 70 Kopplungsagent ohne Haptik

4.4. Zusammenfassung

In diesem Kapitel wurde ein Gesamtkonzept für die Durchführung von Simulationsexperimenten in Hinblick auf interaktive Simulation und Visualisierung in der computergestützten Biochemie vorgestellt. Das Gesamtkonzept besteht dabei aus zwei Teilkonzepten: einem Konzept für den Aspekt der Simulation, einem zweiten für den Aspekt der Interaktion und Visualisierung, die beide, jeweils für sich genommen, bereits tragfähig sind, aber einem gemeinsamen Ansatz entspringen und so in das Gesamtkonzept integrierbar sind.

Das Konzept für die Simulation beinhaltete eine Analyse der Kommunikationsanforderungen der die Simulatoren kapselnden Agenten, der so genannten Kopplungsagenten. Hierzu wurde ein „Dialekt“ der Standard-Agenten-Kommunikationssprache der FIPA in Form einer Auswahl der benötigten *Performatives* erstellt, die *BioSLang*. Dazu passend wurde eine an die Bedürfnisse der computergestützten Biochemie angepasste Inhaltsbeschreibungssprache definiert, die *BiSCeD*.

Das Konzept für die Interaktion und Visualisierung teilt sich wiederum in zwei Konzepte, die aufeinander aufbauen: Die Idee des Kopplungsagenten wird für die Visualisierung und Interaktion als Visualisierungskomponente interpretiert, also eine Softwareeinheit, die ihre Repräsentation unter eigener Verantwortung innerhalb eines Frameworks kontrolliert.

Als Erweiterung für die intuitive Interaktion wird ein neues, multimodales Ein-/ Ausgabegerät vorgeschlagen, die *Virtual Glove Box*, die einem 3-D-Präsentationssystem eine haptische Komponente hinzufügt. Diese haptische Komponente setzt sich auf Seite der Software als Teil des Komponentenkonzepts zur Visualisierung fort.

Die Integration des Visualisierungs- und Interaktionskonzepts in das Konzept des Kopplungsagenten ist aufgrund der Vorüberlegungen nahtlos. Dabei wird nicht vergessen, dass multimodale Displays nicht allgemein verfügbar sind, indem festgestellt wird, dass die haptische Komponente optional ist.

Im folgenden Kapitel wird die Umsetzung des Konzeptes vorgestellt.

5. Realisierung

Die in Kapitel 4 beschriebenen Konzepte sind in einer Reihe von Hard- und vor allem Softwareanwendungen umgesetzt worden. Im folgenden Kapitel sollen diese Realisierungen vorgestellt werden. Zu Beginn wird dabei das Hardwaresystem der *Virtual Glove Box* erläutert, bevor dann im zweiten Unterkapitel die verschiedenen Softwaresysteme beschrieben werden.

Diese Softwaresysteme bestehen zunächst aus dem softwaretechnischen Pendant zur *Virtual Glove Box*, einer Implementierung eines Graphischen User-Interfaces (GUI) mit haptischer Unterstützung, anschließend werden für die drei identifizierten Teilbereiche der computergestützten Biochemie Softwaresysteme vorgestellt. Abschließend werden dann Anwendungen außerhalb der computergestützten Biochemie vorgestellt, in denen Teile des Konzeptes umgesetzt wurden.

5.1. Hardwaresystem

Das vorgestellte Interaktionskonzept nutzt ein neues Ein-/ Ausgabegerät, die *Virtual Glove Box*. Dieses Gerät verbindet

- eine 3-D-Projektion
- mit haptischem Feedback (Force-Feedback) und
- die aus dem Anwendungsbereich entnommene Metapher der Glove Box (siehe Abbildung 71).

Abbildung 71 Reale Glove Box

Um die Metapher der Glove Box in ein Ein-/ Ausgabegerät für Anwendungen in der computergestützten Biochemie, die so genannte Virtual Glove Box, umzusetzen, müssen die wesentlichen Eigenschaften der Ausgangskonfiguration erhalten bleiben. Diese wesentlichen Eigenschaften sind zum einen eine Sichtscheibe, durch die der Arbeitsbereich zu sehen ist, zum anderen die Handschuhanbindung, um in die Kammer zu greifen. In unserer Anordnung gibt es folgerichtig zwei wesentliche Komponenten:

1. das visuelle Display
2. das haptische Display.

Für die visuelle Darstellung der Szene kommt dort, wo bei einer konventionellen Glove Box die Sichtscheibe ist, eine Rückprojektionsscheibe zum Einsatz. Das Bild wird dabei von zwei Projektoren als so genannte passive Stereoprojektion erzeugt. Das Objektiv jedes Projektors ist mit einem Polfilter ausgestattet, so dass unterschiedliche Bilder für jedes Auge projiziert werden können. Der Benutzer trägt zu diesem Zweck eine Brille, die ebenfalls für jedes Auge mit unterschiedlich polarisierten Gläsern ausgestattet ist.

Zur Berechnung des Bildes kommt je nach Konfiguration ein IBM-PC mit einer Wildcat 4210-Graphikkarte (zwei Videoausgänge) oder eine SGI-Onyx zum Einsatz. Die Projektoren sind übereinander angeordnet und projizieren das Bild direkt auf die Projektionsscheibe. Damit ist der Benutzer in die Lage versetzt, das virtuelle Geschehen zu beobachten.

Abbildung 72 Haptisches Exoskelett [161]

Das haptische Display dient nun der direkten manuellen Manipulation der virtuellen Szene. Das Display besteht wiederum aus zwei getrennten Bereichen. Der erste Bereich ist die Dateneingabe, mit deren Hilfe die Bewegungen des Benutzers erfasst werden. Die Finger- und Gelenkstellung des Benutzers wird mit Hilfe von Datenhandschuhen aufgenommen. Gleichzeitig erfasst ein Trackingsystem die Position der Hände im Raum. Aus Fingerstellung und -position lassen sich eine Kollisionsberechnung mit der virtuellen Szene und die graphische Ausgabe der Handposition berechnen. Der zweite Bereich umfasst die

Datenausgabe. Zunächst fällt hierunter die visuelle Ausgabe. Durch die graphische Ausgabe der Handposition kann der Benutzer seine Hände in der virtuellen Szene genau dort sehen, wo er bei einer durchsichtigen Scheibe seine realen Hände sehen würde. Ebenso wichtig ist die haptische Datenausgabe. Zu diesem Zweck kommen zwei Exoskelette vom Type CyberGrasp zum Einsatz [161] (siehe Abbildung 72). Aus den Kollisionsberechnungen der Hände mit der Szene wird über die CyberGrasp-Exoskelette ein haptisches Feedback erzeugt, indem entsprechende Kräfte auf die Finger des Benutzers ausgegeben werden.

Der Benutzer trägt je ein Exoskelett an jeder Hand, das über dem Datenhandschuh getragen wird. Das Tracking der Handposition erfolgt mit Hilfe eines magnetischen Trackers vom Typ Polhemus-FASTRACK. Das Exoskelett mit Datenhandschuh und Receiver für das Tracking wiegt etwa 350 Gramm und wird daher von Benutzern gut toleriert.

Die beiden Aspekte, visuelles Display und haptisches Display, werden in der Virtual Glove Box zusammengeführt, indem direkt unterhalb der Projektionsfläche zwei Öffnungen vorgesehen sind, die es erlauben, in das Gerät zu fassen (siehe Abbildung 68). Gleichzeitig werden so die realen Hände des Benutzers vor seinen Blicken verborgen. Hierdurch wird der Eindruck der Immersion verstärkt.

5.2. Softwaresysteme

Das Schwergewicht dieser Arbeit liegt auf der softwaretechnischen Umsetzung der vorgestellten Konzepte. Im Folgenden werden demnach vier Softwaresysteme umgesetzt:

- die Softwareseite der Virtual Glove Box in Form einer graphischen Benutzungsoberfläche mit haptischer Unterstützung
- eine Molecular-Modeling-Anwendung
- Anwendungen in der Systembiologie
- eine Editorandwendung für Zelldifferenzierungssimulationen.

5.2.1 Realisierung eines graphischen Benutzungsinterfaces mit haptischer Unterstützung

Die konkrete Umsetzung des *EASY*-Konzeptes, also die Umsetzung eines „greifbaren“ Graphischen User-Interfaces (GUI), bedarf einiger komplexer Entscheidungen.

Multithreaded Rendering

Die eingesetzte Grafikkarte ermöglicht, wie oben beschrieben, den parallelen Anschluss der beiden Projektoren. Es stehen zwei Prozessoren zur Verfügung. Die Ansteuerung der Monitorausgänge erfolgt abhängig von der gewählten Auflösung: Ist eine Auflösung von 1280 x 1024 Punkten gewählt, so werden alle Inhalte, welche sich innerhalb dieser Grenzen befinden, auf dem ersten Monitor angezeigt. Alle Inhalte, die vom zweiten dargestellt werden sollen, müssen sich dann im horizontalen Bereich zwischen 1281 und 2560 Punkten befinden. Da die Anwendung bildschirmfüllend laufen soll, ist demnach die Erzeugung zweier Fenster mit den Eckpunkten (0;0),(1280;1024) und (1281;0),(2560;1024) notwendig.

Die Cosmo-API [162] stellt für solche Fälle eine Lösung zur Verfügung, mit deren Hilfe man den beiden Fenstern einen eigenen Thread zuordnen kann. Beiden Threads kann eine Frame-

Funktion übergeben werden, welche bei jedem Loop des Threads aufgerufen wird. Diese kann bei entsprechender Berücksichtigung der Datenintegrität für beide Threads dieselbe sein. So lassen sich die Berechnungen der Simulation in diese übergebene Funktion auslagern und müssen nicht von beiden Threads ausgeführt werden. Das Rendern der Szene, welches für die Erzeugung der 3-D-Ansicht aus verschiedenen Kameraperspektiven erfolgen muss, kann und sollte parallelisiert werden. Es ist nur darauf zu achten, dass die Fenster synchron aktualisiert werden.

Berechnung der Kameraposition

Die visuellen Repräsentationen der virtuellen Hand lassen sich nicht in den Szenegraphen einbinden und benötigen zur korrekten Darstellung einen OpenGL-Kontext. Obwohl Cosmo auf OpenGL basiert, ist der Einsatz zweier Kameras pro Thread notwendig (eine für den Szenegraphen, eine für die virtuellen Hände). Dies erfordert auf der einen Seite die Synchronisation der Kamerapositionen und Blickrichtungen, auf der anderen aber auch die Erzeugung identischer Frustums.

Die Kamerapositionen müssen auch an die Größe des Benutzers anpassbar sein, um einen bestmöglichen Blick auf die Szene sicherzustellen. Zudem ist der Abstand der Kameras der beiden Ansichten variabel zu gestalten, so dass der Augenabstand des Benutzers berücksichtigt werden kann.

Einfügen von Objekten in die Szene

Der Aufbau der Szene ist dem Entwickler einer Simulation nicht abzunehmen. Das Erzeugen der haptischen Repräsentation ist es sehr wohl. Es besteht aus einer Abfolge von immer gleichen Schritten, deren Ergebnis ein Data-Neutral-Knoten ist, welcher die haptische Repräsentation eines Objekts im haptischen Szenegraphen referenziert. Eine Komponente könnte diesen leicht speichern und dem Entwickler einer Simulation durch automatische Updates der beiden Repräsentationen des Objekts viel Arbeit ersparen.

Kollisionsberechnung

Die Collision-Engine läuft ebenfalls als eigener Thread. Um sie einzusetzen, muss die Klasse `vhtSimulation` abgeleitet und die Methode `handleConstraints()` überschrieben werden. Diese Methode wird in jedem Loop des Threads aufgerufen und muss einen Code für den Start der Kollisionsberechnung enthalten. Während der Kollisionsberechnung darf der haptische Szenegraph nicht verändert werden, was durch das Setzen eines Locks realisiert werden kann. Problematisch ist jedoch, dass ohne weiteren Abgleich des Programmablaufs Situationen auftreten können, in denen während eines Loops der Collision-Engine mehrere Loops der Simulation durchlaufen werden können (oder umgekehrt). Der Einsatz eines Locks zur abwechselnden Berechnung von Simulation und Kollisionsdaten ist eine gangbare Lösung für dieses Problem.

Einbinden der Hardware

Die Anpassung der Hardware erfolgt mit Hilfe eines Programms namens DCU, welches die leichte Einrichtung der Kommunikationsparameter ermöglicht. Für jedes angeschlossene Gerät lässt sich ein Name eingeben, über welchen es aus einer Anwendung heraus angesprochen werden kann, solange DCU läuft. So kann mit wenigen Codezeilen die komplette Hardware eingebunden werden.

Einbinden der haptischen Komponenten

Für die Koordination der Komponenten der Simulation ist eine übergeordnete Instanz nötig, welche das Ausführen der Methoden jeder eingebundenen Komponente auslöst. Diese Instanz kann gleichzeitig für das Auswerten von Kollisionspaaren benutzt werden und diese an die

beteiligten Komponenten übermitteln, welche selbst über die Weiterverarbeitung entscheiden können. Eine zentrale Stelle sollte eine Übersicht über die eingebundenen Komponenten verwalten.

Die Hände und die Werkzeuge

Die direkte Manipulation von Objekten ist bereits durch die Benutzung der bereitgestellten Bibliotheken möglich. Für den Einsatz von Werkzeugen muss jedoch ein Verfahren gefunden werden, welches die Kommunikation zwischen jeder Art von Komponenten erlaubt. Wenn ein Hammer auf ein anderes Objekt trifft, reicht es möglicherweise, wenn das Objekt in seiner Liste von Kollisionspaaren den Hammer entdeckt. Wenn ein Pinsel die Farbe eines Objekts verändern soll, reicht die einfache Bestimmung einer Kollision nicht mehr aus. Es ist zusätzlicher Datenaustausch erforderlich.

Implementation

Die Entwicklung der Bibliothek bestand zu einem Großteil aus der Bereitstellung eines Frameworks für den Einsatz der durch die Aufgabenstellung geforderten wiederverwendbaren Oberflächen-Komponenten. Der Vorteil dieser Methode besteht darin, dass ein Entwickler sich allein auf die Definition des Verhaltens der Komponenten beschränken kann. Das entwickelte Grundgerüst einer haptischen Komponente kann zum einen für die Erstellung von Benutzungsoberflächenelementen benutzt werden, zum anderen können andere abstrakte Konzepte für die Erstellung einer Simulation realisiert werden.

Die haptische Komponente

Ziel der Entwicklung war eine Komponente, welche sowohl als Gerüst für ein Werkzeug als auch für eine allgemeine Simulationskomponente eingesetzt werden kann. Hierfür war es nötig, die Gemeinsamkeiten zu finden und in einer Datenstruktur zu vereinen. Die Komponenten sind nur in Verbindung mit dem unten beschriebenen Komponenten-Manager lauffähig.

HapticComponentBase

Die Basis bildet die Klasse `HapticComponentBase`. Sie stellt die folgenden Möglichkeiten bereit:

- Laden beliebiger Geometrien
- automatischer Abgleich von visueller und haptischer Repräsentation
- Speicherung und Abfrage des Data-Neutral-Knotens für den Zugriff auf die haptische Repräsentation
- Speicherung der jeweils aktuellen Kollisionspaare
- zyklisch aufgerufene Methode zur Realisierung eines beliebigen Verhaltens
- Möglichkeit zur Blockierung der Erstellung einer haptischen Repräsentation
- Flags für die Abfrage der Zustände „Gegriffen“, „Nicht Gegriffen“, „Losgelassen“.

Diese Struktur ermöglicht den einfachen Bau eines visuellen Szenegraphen und kapselt die gesamte Aktualisierung und Repräsentation des haptischen Teils der Komponente. Abbildung 73 verdeutlicht dies.

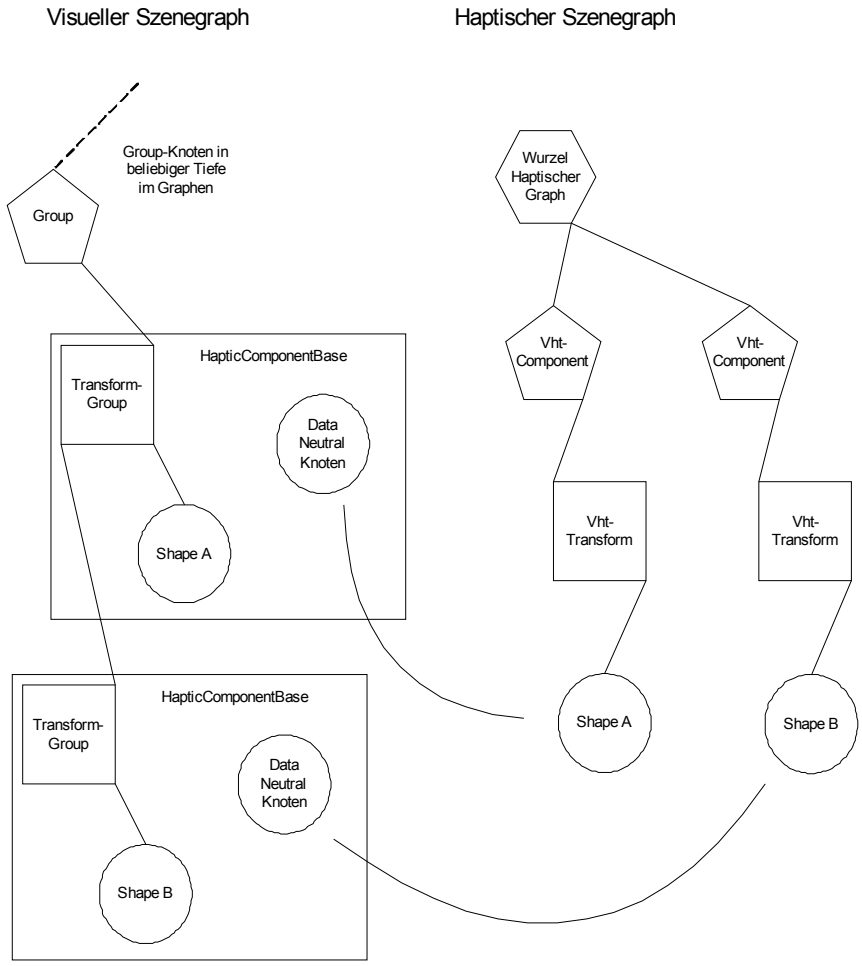


Abbildung 73 HapticComponentBase

Der Szenegraph muss nicht ausschließlich aus haptischen Komponenten bestehen. Er kann den Anforderungen der Anwendung entsprechend konstruiert werden. Nur Objekte, welche direkt von der virtuellen Hand oder indirekt von Werkzeugen manipuliert werden sollen, müssen von diesem Typ sein.

Die Komponente besitzt eine Methode namens `updateLoop(ticks)`, welche in jedem Frame vom Komponenten-Manager aufgerufen wird. Sie enthält drei Abfragen über den aktuellen Zustand, welche als `virtual` deklarierte Methoden aufrufen:

Wurde die Komponente von der virtuellen Hand gegriffen, wird automatisch die im haptischen Szenegraphen geänderte Position mit der im visuellen abgeglichen.

Wurde die Komponente losgelassen, wird ebenfalls die Position aus dem haptischen Szenegraphen übernommen.

Wenn die Komponente nicht gegriffen oder losgelassen wurde, wird die Methode mySimulation(ticks) aufgerufen.

In abgeleiteten Klassen können sie überschrieben werden, um individuelles Verhalten implementieren zu können. Der Parameter `ticks` ist die Zeit in Millisekunden, die für den letzten Simulationsdurchlauf benötigt wurde. Natürlich sind auch Komponenten denkbar,

welche sich nicht bewegen oder sonstige Berechnungen durchführen. Dann kann diese Klasse selbst instantiiert werden.

HapticComponent

Für den Bau eines virtuellen Werkzeugs ist die Klasse `HapticComponentBase` jedoch nicht ausreichend. Ein Werkzeug besteht normalerweise aus einem Griff und einem oder mehreren Teilen zur Manipulation der Werkstücke. Ein Hammer hat einen Stiel und eine spitze und eine flache Seite, um Nägel zu bearbeiten. Die Geometrien der virtuellen Werkzeuge müssen demnach einen Rückschluss darauf zulassen, welches Teil an einer Kollision beteiligt ist.

Die Klasse `Place` dient der Einbindung beliebiger Geometrien zusätzlich zum Hauptobjekt. Der hierfür benötigte Typ von Komponente wurde aus der Klasse `HapticComponentBase` abgeleitet. Es entstand die Klasse `HapticComponent`. Diese besitzt die nötige Logik zur Verwaltung der zusätzlichen Instanzen von `Place`. Damit Place-Instanzen auch kollidieren können, müssen auch sie in den haptischen Szenegraphen eingefügt werden. Auch hier besteht für den Entwickler kein weiterer Handlungsbedarf. Er muss lediglich die Place-Instanzen unter den Root-Knoten der zugehörigen Komponente hängen, das Erzeugen der haptischen Repräsentation erfolgt automatisch bei der Anmeldung am Komponenten-Manager. Abbildung 74 zeigt die Beziehung zwischen visueller und haptischer Repräsentation in den beiden Szenegraphen.

Die Klasse `Place` ist weiterhin dafür gedacht, die haptischen Komponenten um kollidierende Geometrien zu erweitern. So kann die Spitze eines Hammers vom Rest der Geometrie klar unterschieden werden. Ebenso können semantische Constraints durch beliebige Erweiterungen um Place-Instanzen simuliert werden. Place-Instanzen sind nicht greifbar und nur in Verbindung mit der Klasse `HapticComponent` sinnvoll einsetzbar.

Eine visuelle Repräsentation dieser Erweiterungen ist nicht immer sinnvoll, da sie das Erscheinungsbild des eigentlichen Objekts verändert. Für manche Anwendungen kann dies aber durchaus gewünscht sein. Beispielsweise könnte eine Place-Instanz ein Verbindungsstück zwischen zwei Komponenten symbolisieren. Es besteht auf jeden Fall die Möglichkeit, durch Setzen eines Flags die visuelle Repräsentation ebenfalls einzubinden.

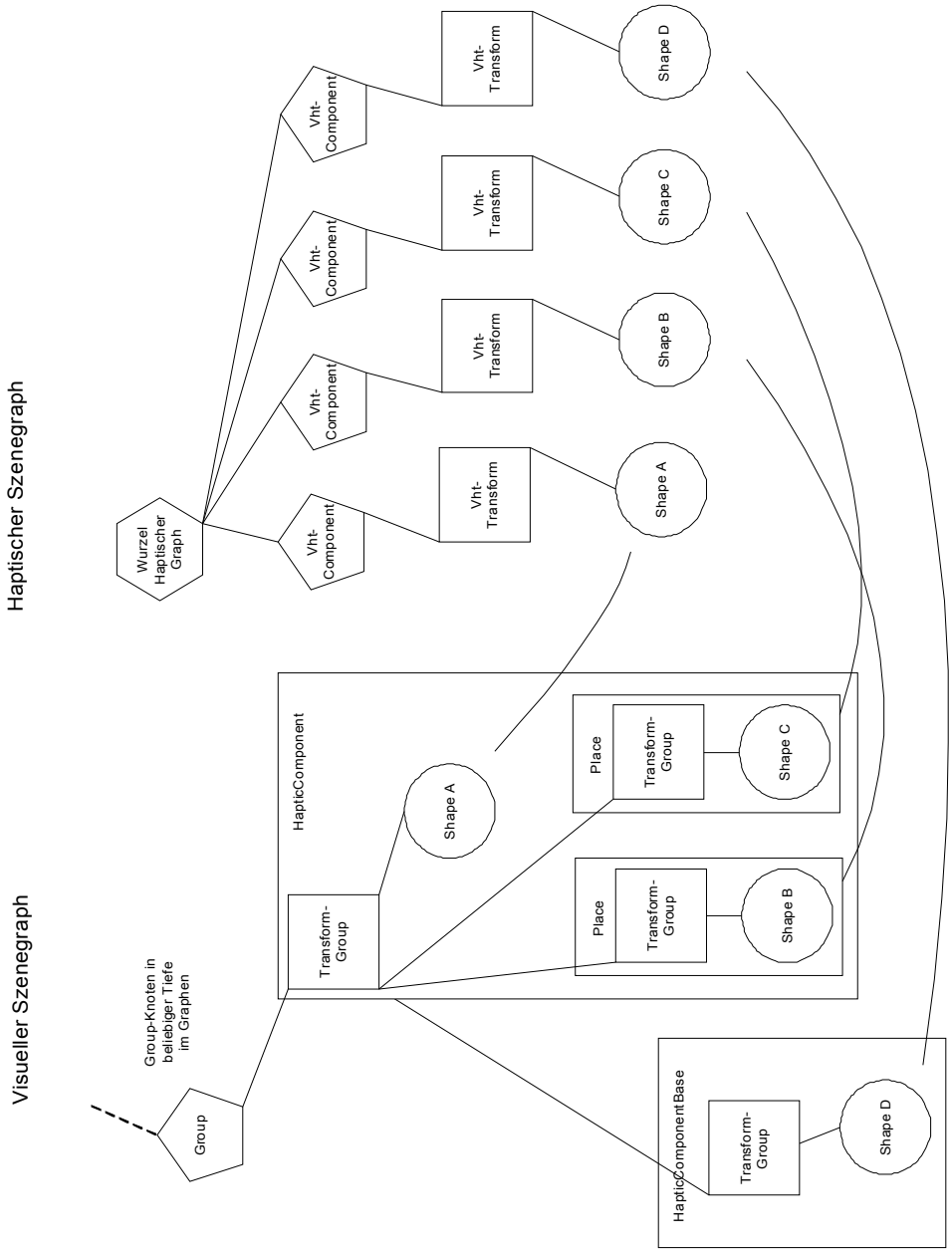


Abbildung 74 Haptische Komponenten

ComponentManager

Die Klasse `ComponentManager` ist der zentrale Punkt des Simulations-Frameworks. Die bereitgestellten Methoden koordinieren den Ablauf von Hand-Interaktion, Kollisionserkennung und Aufruf des *updateLoop* aller registrierten Komponenten.

Registrieren der Komponenten

Jede Komponente muss sich beim Komponenten-Manager anmelden, um im haptischen Szenegraphen registriert zu werden. Durch den Aufruf der Methode

`registerComponent(aComponent)` wird dieser Vorgang gestartet. Die Methode ist überladen und erlaubt das Registrieren von Instanzen der Klassen `HapticComponentBase` und `HapticComponent`. Für alle Geometrien der Komponenten wird zunächst die konvexe Hülle errechnet, anschließend werden diese als haptische Repräsentation der Geometrien in den haptischen Szenegraphen eingefügt. Die registrierte Komponente erhält dabei Zeiger auf die zugehörigen Data-Neutral-Knoten, so dass im weiteren Verlauf der Simulation die einfache Identifizierung gewährleistet ist.

Jede registrierte Komponente wird ebenfalls in eine Liste eingetragen, so dass zu jeder Zeit eine Suche über alle Komponenten durchgeführt werden kann. Ein Werkzeug könnte nach einem passenden Aufbewahrungsort für sich suchen. Auch Objekte, welche keine haptische Repräsentation besitzen sollen, können sich registrieren lassen. Hierfür muss nur eine Komponente ohne bereits zugewiesene Geometrie beim Komponenten-Manager registriert werden. Die visuelle Repräsentation kann später zugefügt und separat in den visuellen Szenegraphen eingebunden werden.

Simulations-Schleife

Die Methode `simulationLoop(ticks)` wird in jedem Frame durchlaufen und koordiniert den Ablauf aller Teile der Simulation, welche mit den Berechnungen im haptischen Szenegraphen zusammenhängen. Die einzelnen Schritte sind folgende:

Alle Komponenten werden auf ihren aktuellen Zustand bezüglich der Interaktion mit den virtuellen Händen geprüft. Das Ergebnis der Berechnung ist später für die Komponenten leicht verfügbar.

Die Liste der Kollisionspaare, welche vorab von der Collision-Engine an den Komponenten-Manager übergeben wurde, wird durchlaufen und zum einen auf sinnlose Kollisionen untersucht (zum Beispiel Kollisionen zwischen einer Instanz vom Typ `Place` und einer Hand), zum anderen werden bei Kollisionen zwischen Hand und haptischen Komponenten die für die Controller-Hardware notwendigen Kontaktflächen berechnet und an die entsprechende Instanz der virtuellen Hand übermittelt.

Die übrigen Kollisionspaare werden an die beteiligten Kollisionspartner, also die Komponenten mit haptischer Repräsentation in der Szene, übermittelt.

Als Letztes folgt der Aufruf der Simulations-Methoden aller registrierten Komponenten in der Reihenfolge ihrer Registrierung.

Haptisches Feedback

Die gesamte Funktionalität für die Vermittlung des haptischen Feedbacks durch das CyberGrasp-Gerät wird von den Fremdbibliotheken bereitgestellt. Es stehen drei Modi zur Verfügung:

- Impedence-Mode: Aus den vom Komponenten-Manager bereitgestellten Kontaktflächen werden ohne weitere Programmierarbeit die Kräfte entsprechend der relativen Position eines Objekts zum berührenden Finger ermittelt und auf die Seilzüge des CyberGrasp übertragen. Sofern keine Berührungen erkannt werden, also keine Kollision zwischen Hand und Objekt, führt das Gerät die Seilzüge entsprechend der Stellung der Finger nach, so dass die Schlaufen am Ende der Seilzüge nicht von den Fingern rutschen.
- Force-Mode: Der Entwickler kann jeden Seilzug direkt ansprechen und, der Anforderung entsprechend, Kräfte auf die Finger übertragen.

- Force-Effect-Mode: Es können beliebige Feedback-Effekte erstellt und gespeichert werden. Beispielsweise können Vibrationen geringer Frequenz oder auch sinusförmige Kraftzuwächse simuliert und ohne weiteren Programmieraufwand an das CyberGrasp übertragen werden.

Für den Einsatz in der Virtual Glove Box ist der Einsatz des Impedence-Modus sinnvoll, da er das Greifen und Fühlen, welches für die direkte Interaktion mit virtuellen Objekten nötig ist, leicht benutzbar bereitstellt.

Um den Gebrauch von Effekten zu ermöglichen, wurde die Übermittlung der Kollisionsdaten an die Hardware in eine eigene Klasse namens `GraspForceManager` gekapselt. Eine gegriffene Komponente kann so den Modus des zur Greifhand gehörigen CyberGrasp ändern und einen Effekt starten. Allerdings ist unklar, wie sich ein Wechsel des Modus während der Laufzeit der Anwendung auswirkt, so dass an dieser Stelle der alleinige Betrieb im Impedence-Modus empfohlen wird. Dies ist auch die Voreinstellung beim Starten des Frameworks.

Programmablauf

Das Computersystem, welches für das Virtual-Glove-Box-Projekt zur Verfügung steht, stellt zwei Prozessoren bereit. Der Ablauf wurde durch Einsatz der Thread-Technologie soweit wie möglich parallelisiert. Dies unterstützt auf der einen Seite das Rendern der beiden Ansichten, auf der anderen Seite kann die aufwendige Kollisionserkennung direkt nach dem Durchlauf der Frame-Funktion erneut gestartet werden. Abbildung 75 zeigt das grundsätzliche Schema der Schleife des Simulations-Frameworks.

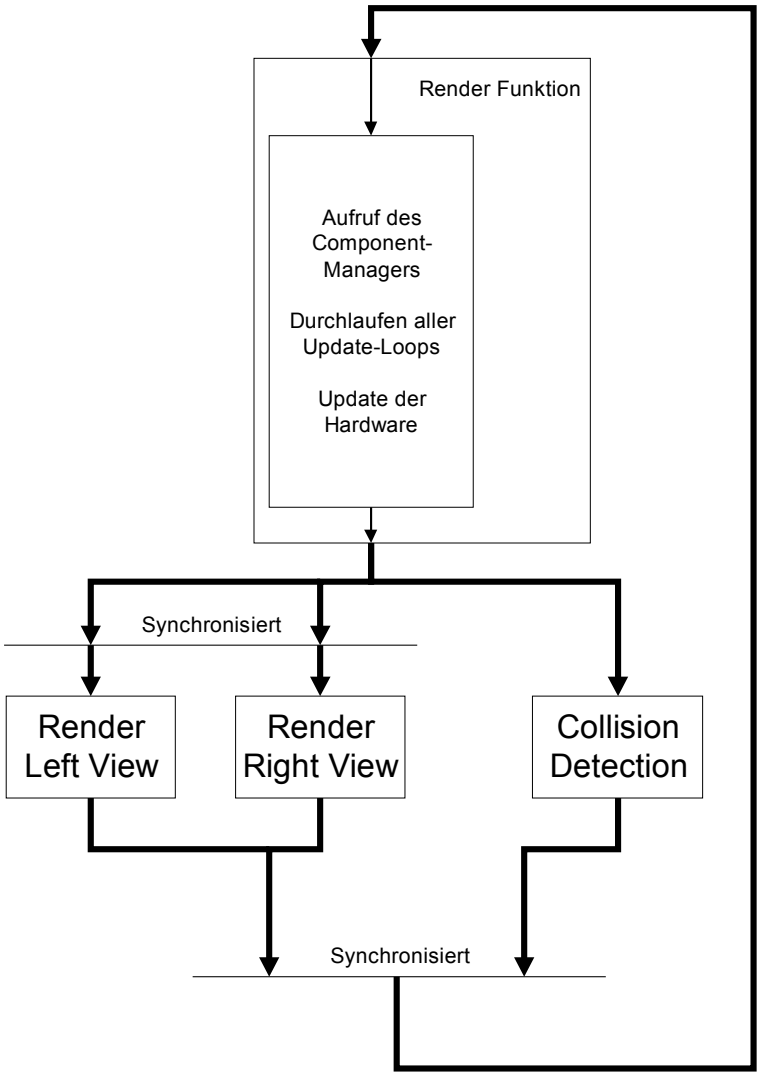


Abbildung 75 Programmablauf

Das Rendern der beiden Ansichten erfolgt erst nach den Berechnungen der Simulation. So ist sichergestellt, dass beide Ansichten die Szene zum gleichen Zeitpunkt darstellen. Das Wechseln der Window-Buffer geschieht ebenfalls synchron, damit beide Augen zur gleichen Zeit die aktualisierte Ansicht der Szene präsentiert bekommen. Die Collision-Engine, welche auch als eigener Thread läuft, kann parallel zum Rendern laufen, da sie selbst nicht auf die visuelle Repräsentation der Szene zugreift.

Der Einsatz von EASY

Um die Anwendung der eben beschriebenen Strukturen zu verdeutlichen, wird an dieser Stelle der Code für den Aufruf des Frameworks und das Einfügen einer haptischen Komponente beschrieben.

```
#include "easyOpenGl.h"


int
main(int argc, char *argv[])


{
// Framework erzeugen durch Instanzierung eines Objekts
// vom Typ easyOpenGl
easyOpenGl *easy = new easyOpenGl;
easy->initClasses();


// eine Komponente erzeugen
HapticComponent *aComponent = new HapticComponent;


// ein Vrml-File Laden
// diese Methode ist durch Cosmo-API vorgegeben
opGenLoader *loader = new opGenLoader;
csTransform *aTransform = new csTransform;
aTransform->addChild(loader->load("einModell.wrl"));


// das Vrml-Modell der Komponente uebergeben
aComponent->addVisual(aTransform);


// Die Komponente registrieren
easy->getComponentManager()->registerComponent(aComponent);


// den Simulations-Loop starten
easy->startLoops();
}
```

Dieser Code führt zu dem in Abbildung 76 dargestellten Ergebnis. Die eingefügte Komponente ist greifbar und verfügt über eine einfache, mit einem 3-D-Modeller erstellte Geometrie. Wie man sehen kann, beschränkt sich die Aufgabe des Entwicklers hauptsächlich auf das Einbinden der Geometrie und das Erstellen der Logik für die eingebundenen Komponenten. Die komplette Dokumentation steht separat im HTML-Format zur Verfügung.

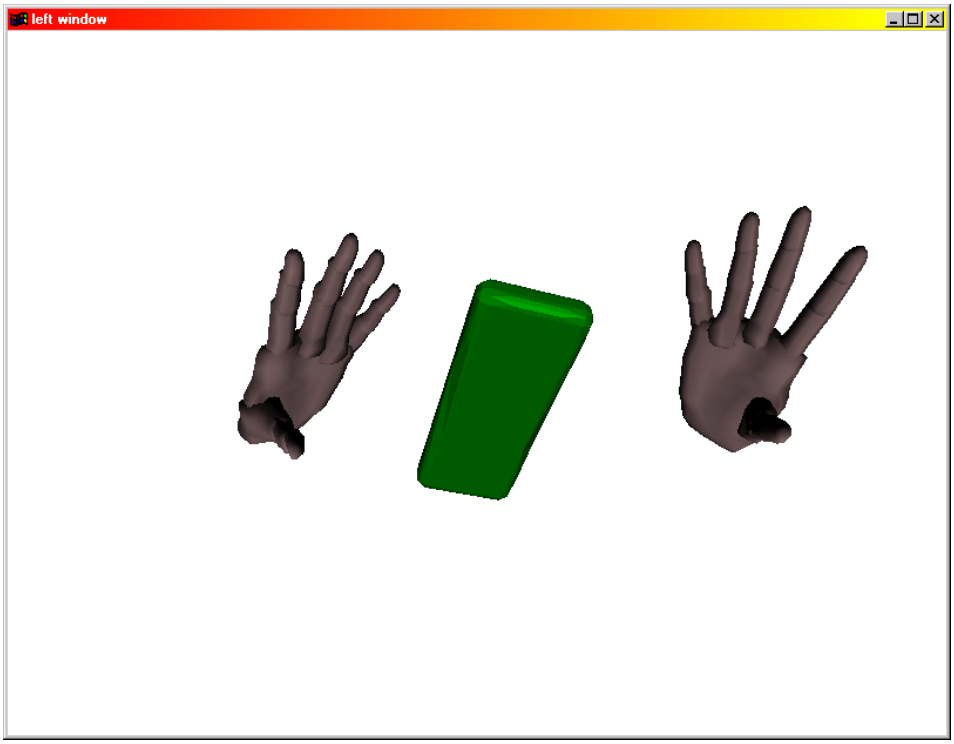


Abbildung 76 EASY-Ergebnis

Wiederverwendbare Benutzungsschnittstellenelemente

Die bereitgestellten Klassen erlauben den individuellen Bau von Schnittstellenelementen. So kann der Entwickler allen Aufgaben, welche nicht durch direkte Interaktion zu lösen sind, ein angepasstes Bedienelement zur Verfügung stellen. Auf diese Weise wird auch die universelle Verwendbarkeit dieser Bibliothek sichergestellt.

Schalter und Regler

Für viele abstrakte Aufgaben bei der Steuerung von Geräten werden Schalter und Regler benutzt. Dies widerspricht zwar der Forderung nach einer kontinuierlichen Bedienung, in Ermangelung einer Alternative ist es nötig, genau diese Elemente zur Verfügung zu stellen. Die Benutzung von Schaltern und Reglern innerhalb dieses Rahmens scheint jedoch deutlich weniger anfällig für Fehlbedienungen zu sein. Zum einen werden die Elemente mit der Hand bedient, über welche man perfekte Kontrolle besitzt, zum anderen können die Elemente an andere Komponenten angehängt werden, wodurch ihr Bezugsrahmen sofort ersichtlich ist.

Schalter

Die Klasse `HapticComponentBase` kann abgeleitet und um Elemente zur Speicherung des Zustands eines Schalters ergänzt werden. Ein erster Ansatz könnte eine Boolsche Variable sein, welche den Zustand Ein/Aus speichert. Problematisch wird es bei der Bedienung des Schalters. Sobald ein Objekt von der virtuellen Hand gegriffen wurde, wird seine Position relativ zur Position der Hand gesetzt. Ein Schalter sollte sich aber nicht aus seinem Rahmen herausbewegen (sofern er einen besitzt). Die Position sollte sich entsprechend der wahrgenommenen Affordances verändern.

Abbildung 77 Lichtschalter

Die Affordances des Schalters aus Abbildung 77 lassen erwarten, dass der Schalter in Form einer Wippe realisiert wurde und offensichtlich der obere Teil der Wippe gedrückt werden muss, um den Schaltzustand zu ändern.

Regler

Für den Bau von Reglern gelten die gleichen Einschränkungen wie für Schalter. Zwar sind auch sie mit Hilfe der Klasse `HapticComponentBase` zu erstellen, doch wie der Schalter sollte der Regler in seiner Bedienung den Erwartungen entsprechen.

Ein Kontrollpanel

Bedienelemente jeder Art könnten aus den bereitgestellten Datenstrukturen gebaut und benutzt werden, sofern haptisches Feedback vermittelt werden kann.

Eine Szene mit mehreren Elementen zur Anwendungssteuerung zeigt Abbildung 78. Es ist eine Desktop-Metapher zu sehen, welche den anderen Elementen als Ablage dient. So wird verhindert, dass Objekte als frei in der Luft hängend wahrgenommen werden. Die Bedienelemente können beliebig angeordnet werden. Auf dem Desktop befindet sich ein Werkzeug, welches jederzeit von der virtuellen Hand gegriffen werden kann. Die „schwebenden“ Moleküle stellen die eigentliche Simulation dar.

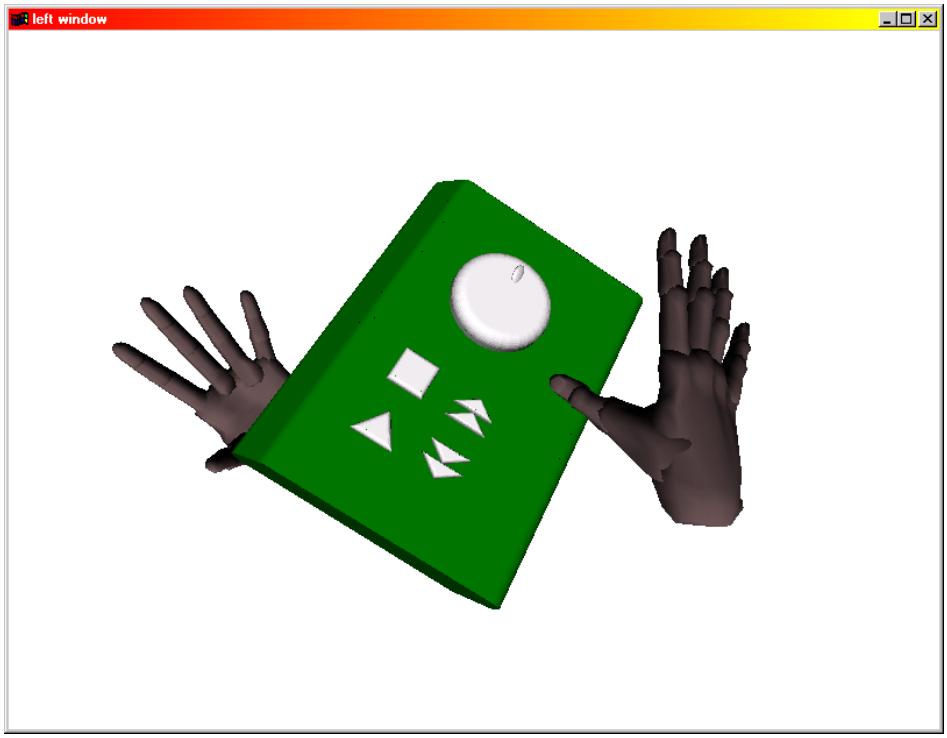


Abbildung 78 Panel

Der Gestaltung der Benutzungsoberfläche sind mit EASY keine Grenzen gesetzt. Die einzigen Beschränkungen liegen in der begrenzten Reichweite der Hände durch das Konstruktionsprinzip der Virtual Glove Box. Eine realistisch wirkende Umgebung ist hier nicht zum Einsatz gekommen. Die einzufügenden Elemente würden die Performanz noch weiter drücken.

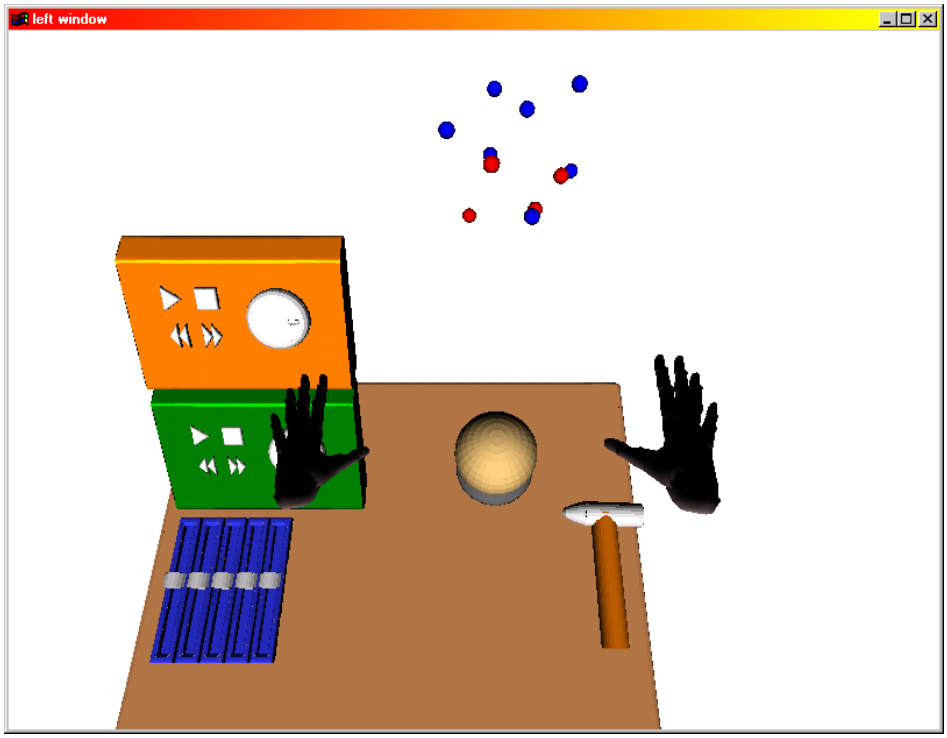


Abbildung 79 Komplexe EASY-Szene

5.2.2 Die Molecular-Modeling-Anwendung

Mit Hilfe der beschriebenen Konzepte und Geräte wurde eine Anwendung aus dem Bereich des Molecular Modeling entwickelt. In diesem Problemkreis versucht der Benutzer, am Rechner neue Moleküle anhand vorgegebener Eigenschaften zu entwickeln oder bekannte Moleküle und Strukturen neu zu kombinieren.

Der Benutzer einer Molecular-Modeling-Anwendung manipuliert die identifizierten Entitäten wie Atome, funktionale (Molekül-) Gruppen und gesamte Moleküle. So wurden Komponenten entwickelt, die entsprechend dem vorgestellten Simulationskonzept Kräfte austauschen und so z. B. Bindungen und Bindungslängen modellieren. Gleichzeitig werden die Atome entsprechend dem Interaktionskonzept „begreifbar“ gemacht, indem ihnen haptische Eigenschaften gegeben werden.

Die damit erstellte 3-D-„Szene“ im Sinne einer Visualisierungsanwendung wird also durch zwei Faktoren gesteuert: zum einen durch die Eingaben, die „Manipulationen“ des Benutzers, zum anderen durch die Simulation.

In diesem Ansatz wird eine typische Aufteilung der Autorenrolle sichtbar, wie sie auch in [163] beschrieben wurde: Der Benutzer interagiert mit den Atomen; diese Bausteine wurden vom Komponentenautor vorgefertigt. Diese Bausteine lassen sich vom Komponentenautor erweitern, zur gleichen Zeit kann der Benutzer durch Kombination der einfachen Bausteine komplexere Komponenten, hier etwa Moleküle, erstellen. Der Komponentenautor ist demnach verantwortlich für die Anbindung der Simulation und die Anbindung der Haptik.

Eine Ansammlung von Bausteinen allein ist nicht ausreichend für eine Applikation, sie ist vielmehr nur ein Teil der Anwendung. Der Rest wird durch den Anwendungsrahmen gebildet. Im Kapitel zur Anforderungsanalyse wurden bereits die Elemente einer „traditionellen" Molecular-Modeling-Anwendung beschrieben. Diese Elemente, jeweils eigenständige Module, waren:

- der 3-D-Editor (ohne Simulationsfunktionen)
- der 3-D-Viewer als Anwendungsmittelpunkt
- verschiedene Simulations- und Evaluierungsmodule
- der Kommunikationsrahmen.

Dagegen wurden in der komponentenbasierten Molecular-Modeling-Implementierung folgende Rollen umgesetzt (siehe Abbildung 80):

- ein Anwendungsframework, das die üblichen Initialisierungsfunktionen übernimmt, wie Initialisierung der Graphikbibliothek (wieder wurde OpenGLOptimizer mit der Cosmo-Szenengraph-API benutzt [161])
- ein Kommunikationsframework für die Komponenten
- ein Komponentenrepository, in dem die vom Komponentenautor entwickelten Atome abgelegt wurden und in dem auch die komplexen Komponenten, wie Moleküle, abgelegt wurden
- ein Editorinterface für Atome, um dem Nutzer die Möglichkeit zu geben, Parameter, wie haptische Eigenschaften, zu konfigurieren

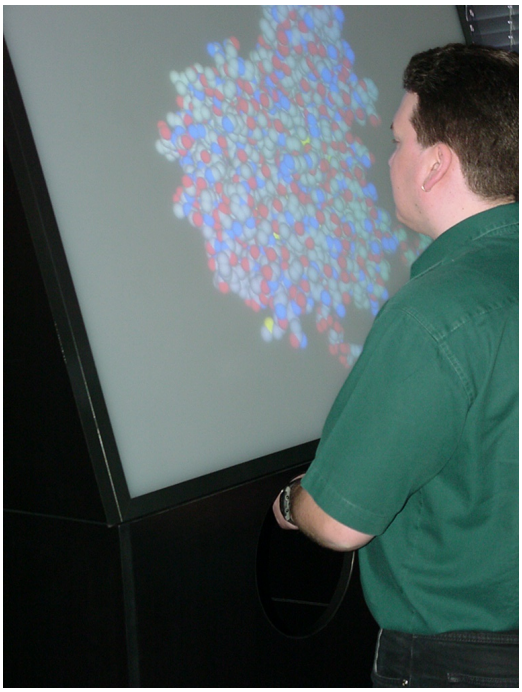


Abbildung 80 Anwendungsbild

Dieser letzte Punkt unterscheidet diese Anwendung besonders von früheren Anwendungen: Es gibt keinen Editor, in dem ein Molekül zusammengesetzt wird. Vielmehr werden Moleküle durch die Eigenschaften ihrer Komponenten, Atome und funktionale Gruppen, gebildet (siehe Abbildung 81). Darüber hinaus bietet jede Komponente einen eigenen Editormodus, wie in der Aufzählung beschrieben.

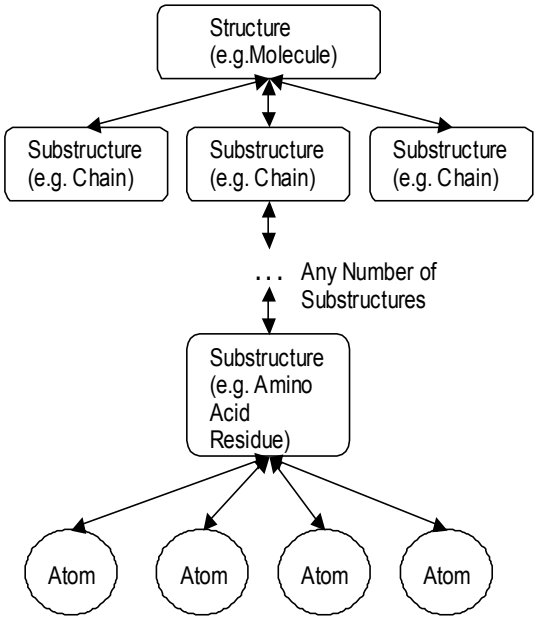


Abbildung 81 Komponentenhierarchie

Das oben erwähnte Komponentenrepository stellt einen flexiblen und erweiterbaren Mechanismus bereit, über den neue Komponenten in die Applikation eingebracht oder bestehende Komponenten erweitert werden können.

Traditionelle Anwendungen erlauben die Erweiterung des Systems, z. B. um neue Simulatorfunktionalität, nur auf anwendungsglobaler Ebene. Dies geschieht etwa durch die Verlinkung neuer Module. Die Anwendung erlaubt durch den Einsatz des komponenteneigenen Editors und des Repository Managers eine feinere Granularität der Erweiterbarkeit.

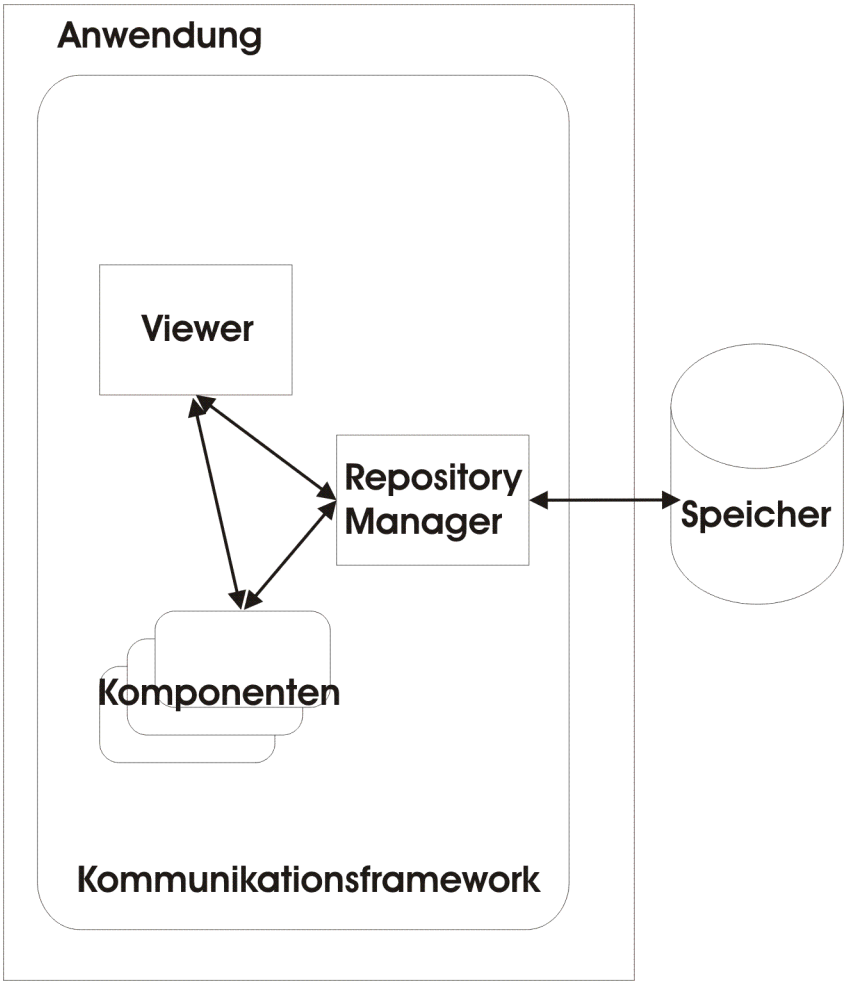


Abbildung 82 Komponenten im Molecular-Modeling-Kontext

Der Repository Manager ist dabei auch als eigenständige Applikation zur Verwaltung der Komponenten zu benutzen. Diese Verwaltungsfunktionen, wie etwa „Suchen“, sind auch aus der Gesamtanwendung heraus nutzbar.

5.2.3 Systembiologieanwendungen

Die vorgestellten Konzepte wurden in drei Anwendungen umgesetzt: einem Zelleditor, einem Simulationsexperimentkonfigurator und einem beispielhaften System gekoppelter Simulatoren. Der Zelleditor erlaubt es, die dreidimensionalen Entitäten innerhalb einer Zelle zu modellieren und so eine dreidimensionale virtuelle Zelle zu erschaffen. Durch spätere Platzierung der simulationsrelevanten Stoffe als Komponenten im Simulationsexperimentkonfigurator können Simulationsexperimente durchgeführt werden. Die gekoppelten Simulatoren demonstrieren, wie die in dieser Arbeit vorgestellten Konzepte für Diffusions-/ Reaktionssimulationen genutzt werden können.

5.2.3.1 Der Zelleditor

Es werden nun die Überlegungen aufgeführt, auf deren Basis die Klassen für das Zellmodell und den Zelleditor entwickelt wurden. Zuerst wird aber der Begriff der Zellmodellierung erläutert.

5.2.3.1.1 Zellmodellierung

Der Prozess der Entwicklung einer dreidimensionalen computergraphischen Repräsentation eines beliebigen physischen Objektes wird gemeinhin als Modellierung bezeichnet. Der hier verwendete Begriff der Zellmodellierung beschreibt den Vorgang der Entwicklung eines 3-D-Computermodells einer Zelle. Er bezieht sich allerdings nicht ausschließlich auf die Entwicklung der 3-D-Repräsentation der Zelle, sondern umfasst auch die semantische Modellierung der Zelle; im Gegensatz zu einem rein graphischen Modell enthält das Modell der Zelle das Wissen darüber, was es darstellt. Die Zelle soll aus mehreren Komponenten modelliert werden können. Dafür wird im folgenden Abschnitt der Begriff des Bausteins eingeführt.

5.2.3.1.1.1 Entitäten der Zellmodellierung

Als Entitäten werden in dieser Anwendung alle Elementarobjekte bezeichnet, die zur Modellierung der Zelle eingesetzt werden können. Jede Entität besitzt eine 3-D-graphische Repräsentation im Modell und kann dort positioniert werden. Entitäten haben einen konkreten Typ, z. B. den Typ Zellkern oder den Typ Mitochondrium, der durch bestimmte Attribute, ein spezifisches Verhalten und einen räumlichen Aufbau beschrieben wird. Das komplette Modell der Zelle wird aus den Entitäten zusammengesetzt. Welche Bausteine für diese Aufgabe zur Verfügung stehen, hängt von den durch den Verwendungszweck der modellierten Zelle gegebenen Erfordernissen ab. Grundsätzlich können beliebige Zellkomponenten durch einen eigenen Baustein-Typ repräsentiert werden, aber auch andere Objekte, z. B. die Membranen oder auch Unterstrukturen der Zellkomponenten, wie der Nucleolus des Zellkerns, könnten als eigenständige Bausteine im Modellierungsprozess eingesetzt werden. Aufgrund der Zergliederung der Zellkomponenten in verschiedene Unterkomponenten – ein Zellkern kann aus zwei Membran-Bausteinen und einem Nucleolus-Baustein zusammengesetzt werden – unterteilt sich die Zelle in mehrere Modellierungsebenen (siehe Abbildung 83 und Abbildung 84).

In der untersten Ebene befindet sich das Modell der Zelle selbst mit den beschriebenen Zellkomponenten als möglichen Bausteinen. Die nächsthöhere Ebene bilden die Modelle der Zellkomponenten mit deren Unterstrukturen als potentiellen Bausteinen. Die Aufspaltung von Bausteinen in weitere Unterstrukturen kann prinzipiell beliebig weitergeführt werden. Jeder Baustein kann somit nicht nur Bestandteil eines bestimmten Modells sein, sondern auch selbst einen eigenständigen Modellierungskontext bilden. Aufgrund dieser Eigenschaft kann die Zelle, wie jede ihrer beschriebenen Unterstrukturen, ebenfalls als Baustein angesehen werden. Diese Annahme wird durch die Tatsache verstärkt, dass Zellen innerhalb eines umfassenderen Modellierungskontextes, z. B. bei der Modellierung eines Organs oder eines Gewebes, selbst die Elementarbausteine sind, aus denen diese Strukturen sich zusammensetzen.

Die komplette Struktur der unterschiedlichen Modellierungsebenen kann in einer hierarchischen Baumansicht abgebildet werden (siehe Abbildung 85). Die Wurzel dieses Baumes bildet das Zellmodell; eine Verbindung zwischen zwei Bausteinen bedeutet, dass der hierarchisch untergeordnete Baustein Teil des Modells des übergeordneten ist. In den Blättern befinden sich diejenigen Objekte, die aufgrund der gegebenen Erfordernisse nicht in weitere Unterbausteine untergliedert werden.

Nachdem nun der Begriff des Bausteins für die Zellmodellierung beschrieben wurde, soll in den nächsten Unterkapiteln versucht werden, die Elemente der Zelle in mehrere Gruppen zu kategorisieren und Bedingungen für die notwendige Bereitstellung eines Zellelements als Modellierungsbaustein aufzuzeigen.

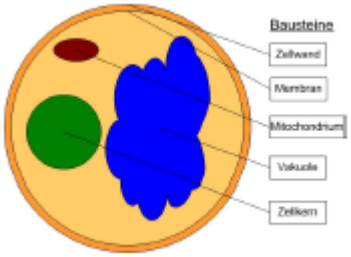


Abbildung 83 Bausteine der Modellierungsebene Zelle

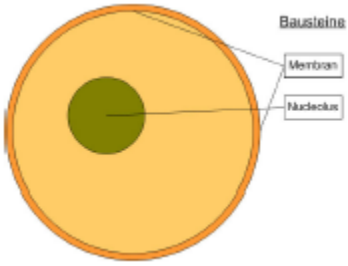


Abbildung 84 Bausteine der Modellierungsebene Zellkern

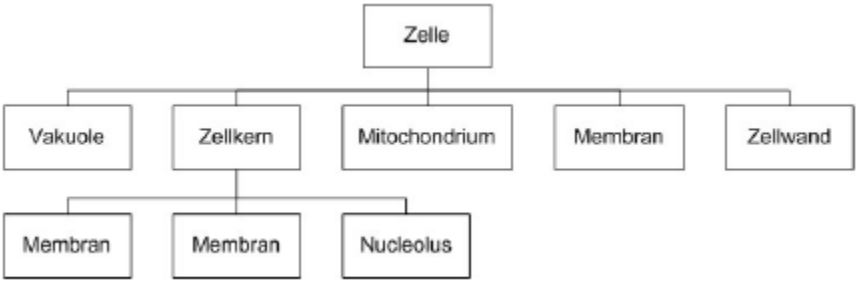


Abbildung 85 Modellierungsebenen der Zelle, dargestellt in einem Baumdiagramm

5.2.3.1.1.2 Klassifikation von Zellstrukturen

Es wird nun versucht, die Entitäten der Zellmodellierung in Hinblick auf ihre bauliche Struktur zu klassifizieren. Die erste klassifizierte Gruppe bilden die Strukturen, die durch eine äußere Hülle in ihrer räumlichen Ausdehnung klar begrenzt werden.

Diese Strukturen besitzen durch die Abgrenzung von der Außenwelt eigenständige Reaktionsräume, sie werden als *begrenzte Strukturen* bezeichnet. Zu diesen Strukturen zählen die membranbegrenzten Zellkomponenten, wie der Zellkern, Mitochondrien, Plastiden, Vakuolen, aber auch membranbegrenzte Vesikel oder Zisternen und die Zelle selbst.

Die Elemente der zweiten Gruppe werden als *begrenzende Strukturen* bezeichnet. Zu dieser Gruppe gehören alle Strukturen, die die Außengrenzen der Strukturen der ersten Gruppe definieren. Hierzu zählen die einzelnen Membranen der Zelle und der Zellkomponenten und die Zellwand.

Im Gegensatz zu den membranbegrenzten Komponenten gibt es mehrere Strukturen, die nicht durch eine äußere Hülle begrenzt werden, die allerdings einen räumlich eng begrenzten Komplex bilden und als funktionale Einheiten beschrieben werden können. Die Strukturen dieser Gruppe werden als *unbegrenzte Strukturen* bezeichnet. Zu dieser Gruppe von Strukturen zählen zum Beispiel die Proteinfilamente des Zytoskeletts oder die Ribosomen; außerdem können in dieser Gruppe Strukturen aufgeführt werden, die aus mehreren in unmittelbarer Beziehung zueinander stehenden Bausteinen zusammengesetzt werden, deren Ausdehnung aber nicht durch eine äußere Hülle bestimmt wird. Dazu zählt z. B. das Dictyosom, das aus einem Zusammenschluss von nah beieinander liegenden membranbegrenzten Zisternen und Vesikeln besteht.

Als letzte Gruppe werden die *komplexen Zellstrukuren* klassifiziert. Als komplexe Strukturen werden Strukturen bezeichnet, für die zwar eine bestimmte Funktionalität beschrieben werden kann, deren räumliche Erfassung allerdings weitaus komplizierter ist, als dies bei den anderen Strukturtypen der Fall ist. Diese Strukturen können sich zum Beispiel über die ganze Zelle erstrecken, wie das Zytoskelett oder das endoplasmatische Retikulum, oder sie sind durch mehrere verteilte Unterstrukturen zusammengesetzt, die nicht in unmittelbarer räumlicher Verbindung zueinander stehen, wie dies beim Golgi-Apparat der Fall ist.

Die Unterscheidung der Bausteine aufgrund ihrer baulichen Struktur ist wichtig in Bezug auf ihr Verhalten innerhalb des Zell-Editors. Die verschiedenen Strukturtypen sind in Abbildung 86 noch einmal zusammengestellt.

Strukturtyp	Beispiel
begrenzt	Zellkern, Mitochondrien, usw.
begrenzend	Membran, Zellwand
unbegrenzt	Dictyosom, Proteinfilamente, Ribosomen
komplex	Zytoskelett, Golgi-Apparat

Abbildung 86 Auflistung der verschiedenen Strukturtypen

5.2.3.1.1.3 Bereitstellung von Bausteinen

In diesem Abschnitt wird kurz erläutert, unter welchen Umständen es sinnvoll ist, eine bestimmte Zellkomponente oder Unterstruktur als eigenen Baustein im Modell zu verwalten. Wie bereits erwähnt, hängt die Tiefe der Untergliederung der Zelle in Unterbausteine von den Anforderungen an die spätere Verwendung des modellierten Objektes ab. Da bislang aber noch keine Abschätzung dieser Anforderungen durchgeführt werden kann, werden ein paar allgemeine Fälle aufgelistet, in denen es notwendig ist, ein Element der Zelle als eigenen Baustein zur Verfügung zu stellen:

- Wenn ein Element der Zelle als eigenes Modell bearbeitet, also aus Unterbausteinen zusammengesetzt werden soll, muss in jedem Fall das Element als Baustein im Zellmodell vorhanden sein.
- Wenn ein Element in seiner semantischen Bedeutung erfasst sein soll, durch ein spezifisches Verhalten und bestimmte Attribute im Modell erkannt werden soll, muss es als eigenständiger Baustein präsent sein.
- Soll ein beliebiges Element im 3-D-Modell transformiert werden können, relativ zu anderen Bausteinen bewegt oder rotiert werden, dann muss es ebenfalls durch einen Baustein repräsentiert werden.

5.2.3.1.2 3-D-Repräsentation der Bausteine

Jeder der im vorigen Kapitel vorgestellten Bausteine besitzt eine dreidimensionale Ansicht im Zellmodell. Das 3-D-Modell der Zelle soll über einen Szenengraphen verwaltet werden. In diesem Abschnitt werden allgemeine Betrachtungen und Vorschläge für die dreidimensionale Darstellung der Bausteine im Szenengraphen zusammengefasst.

Dazu werden zunächst Methoden beschrieben, durch die die Zellbausteine in einem dreidimensionalen Modell dargestellt werden können, und anschließend wird aufgezeigt, wie das Konzept des Szenengraphen zur Lösung der Darstellungsaufgaben eingesetzt werden kann.

5.2.3.1.2.1 Darstellungsmethoden

Um dreidimensionale Objekte im Computer darzustellen, existieren unterschiedliche Modellierungsmethoden. Die 3-D-Repräsentation der Zelle und anderer aus mehreren Unterbausteinen zusammengesetzter Bausteine wird aus den graphischen Darstellungen der Untereinheiten gebildet. Diese Art der Modellierung entspricht den Entitäten, wobei in diesem Fall die Bausteine diese Elementareinheiten darstellen.

Damit für die zusammengesetzten Baustein-Modelle eine 3-D-Darstellung existiert, müssen mindestens die Bausteine, die keine Unterbausteine besitzen, also alle Bausteine, die die Blätter des Modellierungsbaums bilden, über ein eigenes Aussehen verfügen. Um dieses Aussehen zu definieren, wird jeder Entitäten eine Geometrie mit einem Aussehen mitgegeben: Jeder Baustein, der ein eigenes Aussehen besitzen soll, wird aus Elementarobjekten modelliert; die Repräsentation der Elementarobjekte erfolgt durch ein Flächenmodell.

Dadurch ergeben sich bei der Zellmodellierung drei unterschiedliche Ebenen zum Zugriff auf die graphische Darstellung einzelner Bausteine (siehe auch Abbildung 87):

- In der Bausteinebene erfolgt die Modellierung durch Einfügen, Verschieben und Drehen von Baustein-Elementen.
- Die zweite Ebene ist die Ebene der Elementarobjekte der Bausteine. In dieser Ebene werden die Elementarobjekte innerhalb der Baustein-Ansichten parametrisiert, positioniert und orientiert, um das gewünschte Aussehen des Bausteins zu erzeugen.
- In der dritten Ebene kann direkt auf das Geometriemodell einer Entität zugegriffen werden. In dieser Ebene werden Punkte, Linien und Flächen für die elementaren Einheiten bestimmt.

Der Benutzer des Zelleditors wird die Modellierung größtenteils in der Bausteinebene durchführen. Die Modellierung der zweiten und dritten Ebene erfolgt im Allgemeinen durch den Programmierer, den Komponentenautor; trotzdem kann dem Benutzer unter Umständen

auch der Zugriff auf diese Ebenen gestattet werden. Die für die zweite Modellierungsebene benötigten Elementarobjekte werden im nächsten Abschnitt eingehend betrachtet.

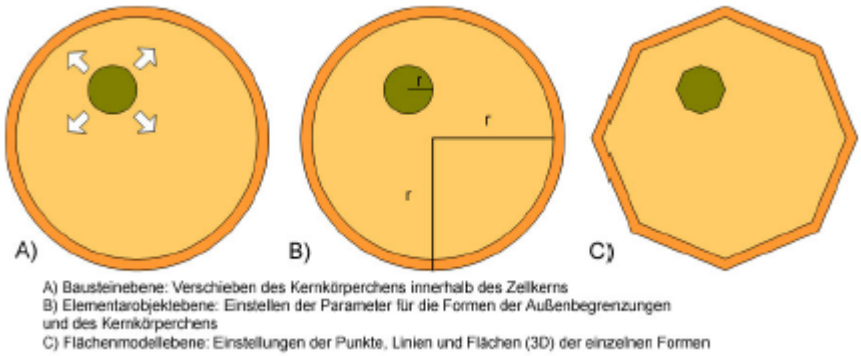


Abbildung 87 Darstellungsebenen des Zellkerns

5.2.3.1.2.2 Elementarobjekte

Ein Elementarobjekt ist eine durch verschiedene Parameter eindeutig definierte Form, die bei der Entwicklung der 3-D-Ansichten der Bausteine eingesetzt werden kann.

Bei üblichen geometrischen Formen, wie bei einer Kugel oder einem Zylinder, sind die zu definierenden Parameter klar: Eine Kugel wird durch einen Radius, ein Zylinder durch Radius und Höhe beschrieben. Bei der Implementierung von Elementarkörpern sind allerdings keine Grenzen gesetzt, die Formen können beliebig komplex sein, und selbst wenn für sie keine logischen Parameter mehr formuliert werden können, sollten zumindest Skalierungsparameter für die drei Raumachsen spezifiziert werden. Zur Beschreibung der Geometrie der Elementarobjekte wird ein Flächenmodell genutzt.

Damit dieses Geometriemodell für alle Objekte einheitlich aufgebaut ist, gelten für die Beschreibung folgende Richtlinien:

- Das Flächenmodell besteht ausschließlich aus den üblichen Graphikprimitiven, wie Dreiecksflächen oder Quads.
- Alle Punkte des Modells werden gemeinsam gespeichert.
- Damit eine Textur über das Objekt gelegt werden kann, muss eine entsprechende Textur-Kachelung der Flächen des Modells spezifiziert werden.

5.2.3.1.2.3 Der Szenengraph

Die graphische 3-D-Darstellung des Modells wird in einem Szenengraphen gehalten. In dieser Anwendung wird die OpenSceneGraph-Szenengraph-Bibliothek benutzt [164]. Jeder Baustein des Modells besitzt eine eigene 3-D-Repräsentation, die in einem Teilbereich des Graphen untergebracht ist. Der Aufbau des Teilgraphen der Bausteine und die Zusammensetzung des gesamten Szenengraphen werden in diesem Abschnitt beschrieben.

Die hierarchische Struktur des Zellmodells muss auch durch den Szenengraphen widergespiegelt werden. Jeder Teilgraph eines Bausteins wird durch einen eigenen Gruppenknoten aufgespannt. Damit die Bausteine im Modell bewegt werden können, wird als

Basisknoten der Bausteine ein Transformationsknoten mit Gruppierungsfunktion verwendet, der für jeden Baustein ein lokales Koordinatensystem definiert und es ermöglicht, Transformationen auf den kompletten Teilgraphen des Bausteins auszuführen. In Abbildung 88 wird der Aufbau des Szenengraphen des in Abbildung 85 gezeigten Modellbaumes dargestellt.

Die Vater-Sohn-Beziehungen aus der Modellierungshierarchie der Bausteine werden nun auch im Szenengraphen nachgebildet. Für jeden Baustein werden die Transformationsknoten von untergeordneten Bausteinen innerhalb des eigenen Transformationsknotens gruppiert. Dadurch übernimmt der Baustein die 3-D-Darstellung seiner Unterbausteine in die eigene Ansicht.

Neben der Möglichkeit der Inbesitznahme fremder Ansichten untergeordneter Bausteine muss ein Baustein auch eine eigene Ansicht implementieren können; in jedem Fall gilt das für die Bausteine, die keine Untereinheiten besitzen. Um die eigene Ansicht von der durch die Unterkomponenten definierte Ansicht zu trennen, wird für jeden Baustein unterhalb des Transformationsknoten ein weiterer Gruppierungsknoten eingefügt.

Die geometrischen Informationen der Elementarobjekte werden in einem Geometrieknoten untergebracht, der sich wiederum in einem Transformationsknoten befindet. Um ein Elementarobjekt in die Bausteinansicht einzufügen, wird der Transformationsknoten des Elementarobjekts in den Gruppenknoten des Bausteins eingefügt.

Durch die Kapselung des Geometrieknotens unter einen zusätzlichen Transformationsknoten kann das Elementarobjekt beliebig im Baustein-Modell transformiert werden. Der Aufbau des Szenen-Teilgraphen eines Bausteins ist in Abbildung 89 aufgeführt.

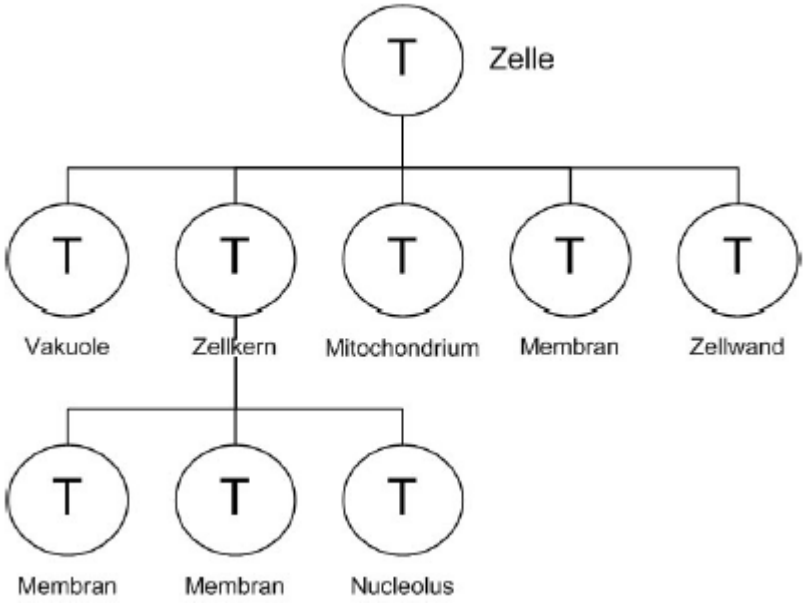


Abbildung 88 Aufbau des Szenengraphen des Modells

Durch den bisherigen Aufbau des Szenengraphen werden lediglich die Geometrien der Bausteine beschrieben. Der Aufbau wird im nächsten Abschnitt um Material und Darstellungseigenschaften erweitert.

5.2.3.1.2.4 Material- und Darstellungseigenschaften

Die Bausteine besitzen nicht nur ein geometrisches Aussehen, sondern auch bestimmte Material- und Darstellungseigenschaften. Normalerweise übernehmen die Bausteine die Darstellungseigenschaften von den Elementarobjekten. Diese Objekte können unterschiedliche Eigenschaften besitzen, in manchen Fällen ist es allerdings erwünscht, für alle Elementarobjekte eines Bausteins oder sogar für die gesamten Ansichten aller Unterbausteine dieselben Material- und Darstellungseinstellungen anzuwenden.

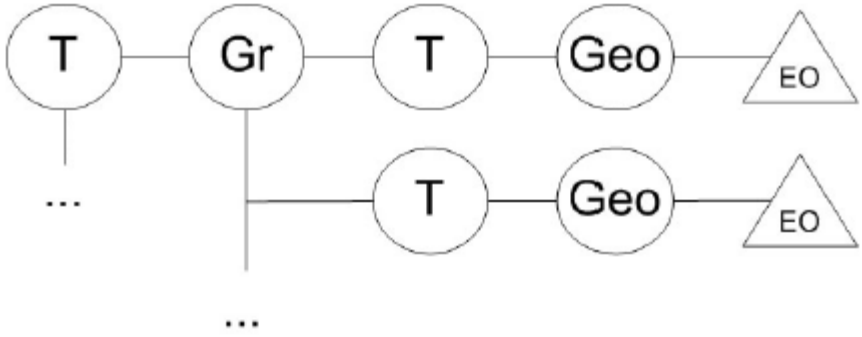


Abbildung 89 Aufbau des Szenengraphen eines Bausteins

Den Knoten des Szenengraphen können Eigenschaftszustände zugeordnet werden. In der Regel werden Eigenschaften untergeordneter Knoten von gleichen Eigenschaften übergeordneter Knoten überschrieben. Innerhalb des Szenengraphen eines Bausteins gibt es drei Positionen, an denen die Eigenschaftsänderungen durchgeführt werden können (siehe Abbildung 90):

- Eigenschaftszustände, die dem Basis-Transformationsknoten des Bausteins zugewiesen werden, gelten für den eigenen Gruppenknoten und alle Transformationsknoten untergeordneter Bausteine. Bei einer solchen Zuweisung ändert sich also die komplette 3-D-Darstellung eines Bausteins. Diese Einstellungen werden überschrieben, wenn ein beliebiger hierarchisch übergeordneter Baustein die entsprechenden Eigenschaften innerhalb seines Transformationsknotens setzt.
- Wenn die Eigenschaftsänderungen dem Gruppenknoten des Bausteins zugewiesen werden, dann ändert sich die Darstellung aller eigenen Elementarobjekte. Durch die Änderung wird nur die individuell durch den Baustein definierte Ansicht betroffen, nicht aber die Darstellung von potentiellen Unterbausteinen.
- Für jedes Elementarobjekt können Eigenschaften gesetzt werden, die dann ausschließlich für dieses Objekt gelten, allerdings nur dann in der Ansicht übernommen werden, wenn kein übergeordneter Knoten die vorgenommenen Einstellungen überschreibt.

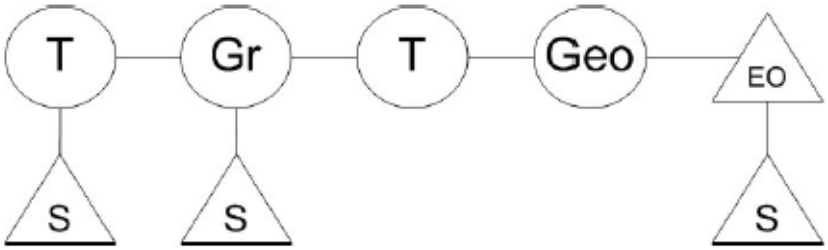


Abbildung 90 Aufbau des Szenengraphen eines Bausteins mit Material-und Darstellungseigenschaften

5.2.3.1.3 Modellierungsrichtlinien

Bei der Modellierung eines Bausteins gelten bestimmte Regeln, aus welchen Bausteinen er zusammengesetzt werden kann. Ein Zellkern wird z. B. aus zwei Membranen und einem Nucleolus gebildet, besitzt allerdings keine Mitochondrien oder Vakuolen.

Wird bei der Modellierung gegen diese Regeln verstoßen, entspricht der modellierte Baustein im Aufbau nicht mehr seinem logischen Typ. Neben den Regeln über die Zusammensetzung eines Bausteins können eventuell noch weitere Richtlinien aufgestellt werden, z. B. über Form oder Größe oder Volumen eines Bausteins.

Prinzipiell ist der Benutzer für die Korrektheit der modellierten Objekte verantwortlich. Modellierungsregeln, die sich abstrakt formulieren lassen, können allerdings in die Programmlogik, gekapselt im zuständigen Kopplungsagenten, eingebunden werden, so dass ein Modellierungstool durch Auswertung dieser Regeln dem Benutzer Hilfestellungen anbieten und ihn bei der Modellierung in die richtigen Bahnen lenken kann. Hierbei ist es wichtig, dem Benutzer notwendige Schranken zu setzen, ihm auf der anderen Seite aber noch genügend Freiheiten zur individuellen Gestaltung zu bieten.

5.2.3.1.4 Modellierung unterschiedlicher Zelltypen

Bei der Modellierung einer Zelle muss der Typ der zu modellierenden Zelle berücksichtigt werden. Zellen unterschiedlichen Zelltyps sind verschieden groß, besitzen andere Formen und beherbergen unterschiedliche Strukturen. Komponenten eines bestimmten Typs können ebenso verschiedene Ausprägungen in unterschiedlichen Zelltypen besitzen.

Eine Konsequenz dieser Typisierung der Zelle ist, dass die im vorherigen Abschnitt beschriebenen Modellierungsrichtlinien zelltypabhängig formuliert werden müssen. In Abbildung 91 ist ein Ausschnitt eines Kategorisierungsbaums dargestellt. Je genauer der Typ der zu modellierenden Zelle klassifiziert wird, umso präziser können die Richtlinien formuliert werden. Besteht die Modellierungsaufgabe darin, eine beliebige Zelle zu erstellen, so muss dem Benutzer bei der Modellierung ein größerer Spielraum geboten werden, als wenn das zu modellierende Objekt eine Zelle vom Typ Saccharomyces cerevisiae oder eine menschliche Muskelzelle darstellen soll, über deren Aufbau sehr viel präzisere Aussagen zusammengestellt werden können.

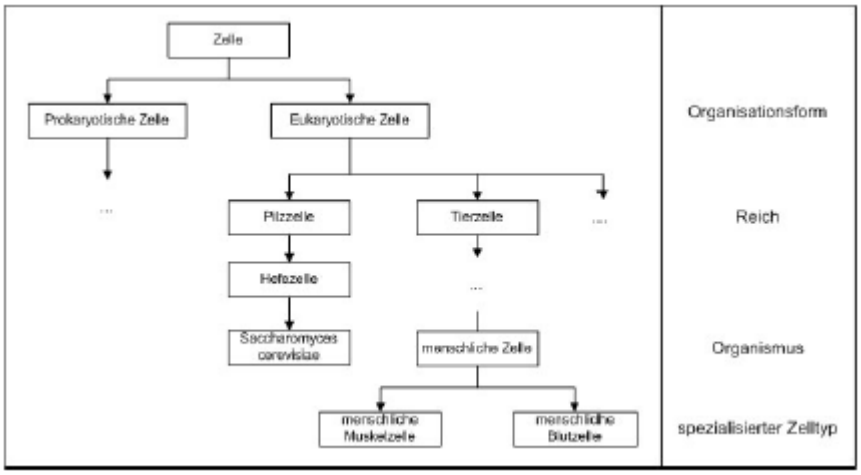


Abbildung 91 Ausschnitt eines Klassifizierungbaumes von Zellen

5.2.3.1.5 Die Klassen des Zellmodells

Ausgehend von Betrachtungen des Zellmodells in den vorigen Abschnitten, wurden mehrere Klassen entwickelt, die im Folgenden kurz vorgestellt werden. Jeder Bausteintyp wird durch eine eigene Klasse repräsentiert, die das Verhalten, die Attribute und die 3-D-Grafik des Bausteins beinhaltet.

Als Basisklasse der spezialisierten Bausteinklassen wurde die Klasse *Component* entworfen. Für Bausteine mit begrenzender Struktur wird zusätzlich eine Klasse *Border* zur Verfügung gestellt, die die Component-Klasse um für diesen Strukturtyp spezifische Elemente erweitert.

Die graphischen Eigenschaften und die Schnittstellen zur Manipulation der Graphik der Bausteine werden in einer speziellen Klasse *View* gekapselt. Durch die Klasse *Shape* können bei der Baustein-Modellierung eingesetzte Elementarobjekte, parametrisierbare Formen oder Körper, beschrieben werden.

Zur Einführung der vorgestellten Modellierungsrichtlinien in das Zellmodell wurde die Klasse *ComponentManager* entwickelt. Spezielle Richtlinien für *Border*-Objekte werden in der von der *ComponentManager*-Klasse erweiterten Klasse *BorderManager* verwaltet. Um die Klassifizierung der Zellen im Modell berücksichtigen zu können, werden die Klassen *Category* und *Categorisation* bereitgestellt. Eine Klasse *ComponentVisitor* dient dazu, klassifizierungsabhängige Richtlinien für beliebige Bausteine aufstellen zu können. Der Aufbau der soeben vorgestellten Klassen wird in den folgenden Abschnitten genauer erläutert.

5.2.3.1.5.1 Bausteinklassen

In diesem Abschnitt werden die Klassen vorgestellt, die zur Repräsentation der im vorherigen Abschnitt beschriebenen Bausteine entworfen wurden.

Die Klasse Component

Die Klasse Component ist die Basisklasse aller Bausteine, das heißt, jeder Baustein wird durch eine eigene Klasse repräsentiert, die das Grundgerüst von der Component-Klasse erbt. Es werden nun einige Eigenschaften und Funktionen der Klasse Component aufgelistet:

Jeder Baustein enthält den Namen seiner konkreten Bausteinklasse und kann dadurch im System eindeutig als Objekt einer bestimmten Klasse identifiziert werden.

- Für jeden Baustein wird in der Klasse Component einer der im vorherigen Abschnitt beschriebenen Strukturtypen festgelegt.
- Die Component-Klasse dient als Containerklasse für Component- und Border-Objekte. Sie kann diese Objekttypen in sich aufnehmen und verwalten. Die Besitzverhältnisse bilden die Vater-Sohn-Beziehungen der Modellhierarchie nach. Nur Bausteine mit begrenztem Strukturtyp können Border-Objekte aufnehmen.
- Jedes Baustein-Objekt besitzt ein Objekt der View-Klasse, in dem die graphischen Eigenschaften verwaltet werden.
- Jedes Baustein-Objekt besitzt ein Objekt der Klasse ComponentManager, das die Modellierungsrichtlinien definiert.
- Für jede Component-Klasse wird eine virtuelle Clone-Funktion definiert. Jede spezialisierte Bausteinklasse muss diese Funktion und einen geeigneten Copy-Konstruktor implementieren, damit geklonte Abbilder des Bausteins erstellt werden können.
- Damit Bausteine gespeichert und gespeicherte Bausteine geladen werden können, muss jeder Baustein entsprechende Serialize-/ Unserialize-Funktionen implementieren.

Die Klasse Border

Von den vier verschiedenen Strukturtypen werden nur die begrenzenden Komponenten durch eine eigene Klasse repräsentiert. Die Klasse Border erweitert die Component-Klasse um Eigenarten, die für alle begrenzenden Komponenten gelten:

- Eine Komponente kann mehrere Begrenzungen besitzen, der Zellkern z. B. besitzt zwei Membranen. Diese Begrenzungen sind schichtweise übereinander angeordnet, die verschiedenen Schichten können, beginnend von der innersten Schicht, nach außen hin indiziert werden. Der entsprechende Index – es wird, ausgehend von der innersten Schicht (Index 1), nach außen hin inkrementiert – wird in der Klasse Border gespeichert.
- Die 3-D-Geometrie einer Begrenzung ist durch eine in sich geschlossene Fläche, z. B. einer Kugel, gegeben. Eine solche Fläche kann genau durch ein Shape-Objekt beschrieben werden. Jedes Border-Objekt besitzt genau ein solches die Begrenzung definierendes Shape-Objekt.
- Die Border-Bausteine speichern Zeiger auf existierende Border-Objekte ihrer Nachbarschichten.

5.2.3.1.5.2 Klassen zum Beschreiben des graphischen Auftretens

Das oben beschriebene Konzept für die 3-D-Darstellung der Zellbausteine wird durch die Klassen *Shape* und *View* realisiert. Die Zusammenhänge zwischen den beiden im Folgenden

beschriebenen Klassen zur graphischen Darstellung und den im vorigen Abschnitt beschriebenen Baustein-Klassen wird in Abbildung 92 in Form eines Klassendiagramms dargestellt.

Die Klasse Shape

Ein Shape-Objekt beschreibt eine durch Parameter definierbare geometrische Form. Shape-Objekte werden als Elementareinheiten bei der Modellierung der 3-D-Darstellung der Bausteine verwendet. Die Klasse Shape ist eine Spezialisierung der OpenSceneGraph-Geometry-Klasse und kann dadurch direkt in einen OpenSceneGraph-Szenegraphen eingefügt werden. Bei der Darstellung der Geometrie wurde auf die oben beschriebenen Richtlinien geachtet. Spezielle Eigenschaften und Funktionen der Shape-Klasse sind:

- Jedes Shape-Objekt enthält den Namen seiner konkreten Bausteinklasse und kann dadurch im System eindeutig als Objekt einer bestimmten Shape-Klasse identifiziert werden.
- Einem Shape-Objekt kann eine Farbe und eine Textur zugewiesen werden, die allerdings nur dargestellt wird, wenn übergeordnete Knoten die Einstellungen nicht überschreiben.
- Wie zu Beginn des Abschnittes erwähnt, definiert jede Shape-Klasse bestimmte Parameter. Die Anzahl und der Typ der Parameter hängen von der Form ab, die durch die Shape-Klasse beschrieben werden soll. Damit auf die Parameter von Shape-Objekten zugegriffen werden kann, ohne dass die konkrete Klasse bekannt ist, werden die Parameter nicht als klasseneigene Attribute deklariert, sondern in einer Liste gespeichert und über Identifikationsnummern oder Namen angesprochen.
- Bei Geometry-Objekten werden normalerweise nur die Außenseiten der Flächen gezeichnet, d. h. nur die Seiten, in deren Richtung der über die Normalvektoren der Eckpunkte approximierte Flächennormalvektor zeigt. In einigen Fällen müssen bei einem Geometry-Objekt auch die Innenseiten sichtbar sein; dies ist z. B. bei Shape-Objekten der Fall, die die 3-D-Geometrie von Border-Bausteinen definieren. Die Shape-Klasse unterstützt die Möglichkeit, die zweiseitige Darstellung eines Objektes zu erzwingen.
- Von der Punktedichte des Oberflächennetzes des Flächenmodells hängt die Genauigkeit der Darstellung des durch das Modell beschriebenen dreidimensionalen Objektes ab. Ein dichtes Netz bietet eine feinere Darstellung, wobei sich Speicherbedarf und Rechenkapazität für die Rendering-Operationen erhöhen. Umgekehrt wird durch ein grobes Netzwerk eine ungenauere Darstellung, aber auch erhöhte Performance erzielt. Durch Beleuchtungstechniken, wie z. B. das Gouraud-Shading-Verfahren, und den Einsatz von Texturen kann das Ergebnis bei gröberer Detailierung positiv beeinflusst werden. Es muss ein geeigneter Kompromiss zwischen Performance und Detailstufe gefunden werden. Shape-Objekte besitzen einen Detail-Parameter, der die Punktedichte für das darzustellende Objekt definiert.
- Jede konkrete Shape-Klasse muss eine Init-Funktion neu implementieren. Die Init-Funktion wird bei der Neuinitialisierung der 3-D-Geometrie eines Shape-Objektes und bei der Änderung von Parametern aufgerufen.

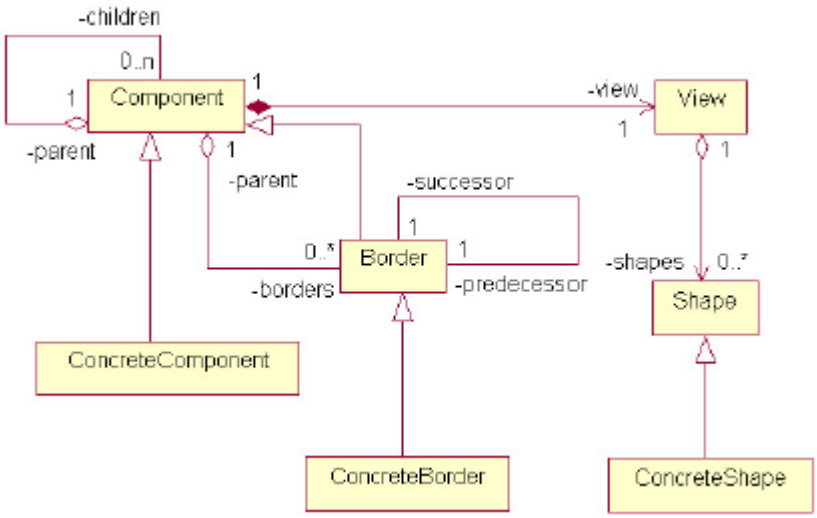


Abbildung 92 Klassendiagramm zum Verdeutlichen der Zusammenhänge zwischen Baustein-, View- und Shape-Klassen

Die Klasse View

Die Klasse View verwaltet das graphische Aussehen eines Bausteins. In dieser Klasse wird der Teilbaum des Bausteins aus dem Szenengraphen des Gesamtmodells gespeichert und verwaltet. Zu diesem Zweck enthält ein View-Objekt Verzeigerungen auf

- den Transformationsknoten, der den Teilgraphen aufspannt
- den Gruppenknoten, der die bausteineigene 3-D-Darstellung enthält
- alle Shape-Objekte, die die 3-D-Geometrie beschreiben.

Für den Zugriff auf den Teilbaum des Szenengraphen werden folgende Schnittstellen bereitgestellt:

- Methoden zum Hinzufügen, Entfernen, Verschieben und Drehen von Shape-Objekten
- Methoden zum Verschieben und Drehen des durch den gesamten Teilgraphen beschriebenen 3-D-Objektes
- Methoden zum Verlinken des Teilgraphen eines anderen Views mit dem eigenen Teilgraphen und die entsprechenden Entkopplungsoperationen. Ein Teilgraph eines Bausteins wird genau dann Teil des Teilgraphen eines anderen Bausteins, wenn der eine Baustein in den Besitz des anderen Bausteins übergeht.
- Methoden zum Setzen von Farb- und Attributwerten für den Transformations- oder den Gruppierungsknoten.

5.2.3.1.5.3 Managerklassen

Es werden in diesem Abschnitt die Klassen vorgestellt, die zur Beschreibung der Modellierungsrichtlinien entwickelt wurden.

Die Klasse ComponentManager

Jedes Baustein-Objekt besitzt einen eigenen ComponentManager, in dem die Modellierungsregeln für das Objekt festgelegt werden. Der Basis-ComponentManager legt für ein Objekt drei verschiedene Richtlinien fest:

- Eine Liste von Objekten der Klasse *ComponentRestriction* definiert, welche und wie viele Bausteine zur Modellierung des Component-Objektes verwendet werden dürfen. Eine *ComponentRestriction* enthält folgende Angaben: den Namen des Bausteintyps, einen Wert für die minimal erlaubte Anzahl, einen Wert für die maximal erlaubte Anzahl, einen Zähler, der angibt, wie viele Objekte des Bausteins bereits vorhanden sind. Ein Baustein kann dem Objekt nur hinzugefügt werden, wenn eine entsprechende *ComponentRestriction* vorhanden ist und die maximale Komponentenanzahl noch nicht erreicht wurde. Die Durchschnitts- und Minimalwerte können innerhalb der Programmlogik des Editors für unterstützende Funktionen eingesetzt werden.
- Eine Liste von Objekten der Klasse *BorderRestriction* definiert, welche Border-Bausteine zur Modellierung des *Component-Objektes* verwendet werden dürfen. Eine *BorderRestriction* enthält folgende Angaben: den Namen des Bausteins, einen Wert für den Index der durch den Baustein definierten Border-Schicht, ein Flag, das gesetzt wird, falls ein Objekt dieses Bausteins bereits in der angegebenen Schicht eingefügt wurde.
- Ein Objekt der Klasse *ViewRestriction* legt Farb- und Textureigenschaften fest. Sind diese Eigenschaften gesetzt, dann soll dies andeuten, dass für alle Shape-Objekte und Unterbausteine eine einheitliche Farbe bzw. Textur gelten soll.

In Abbildung 93 a) werden die Modellierungsrichtlinien für einen normalen Zellkern aufgelistet.

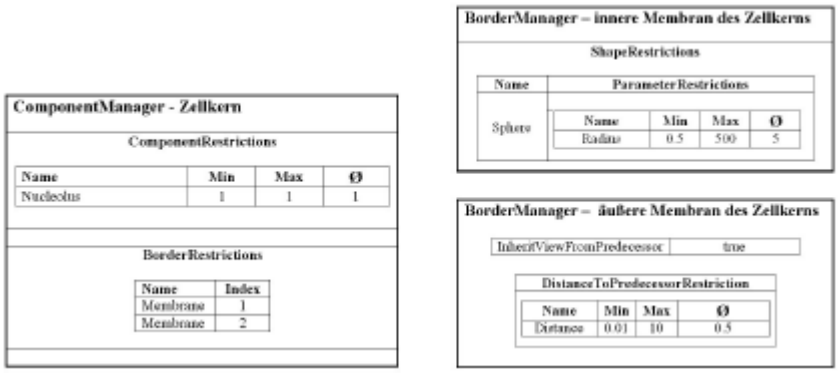


Abbildung 93 a) ComponentManager eines Zellkerns mit einem Kernkörperchen und einer Doppelmembran
b) BorderManager der Membranen eines kugelförmigen Zellkerns

Für die Entwicklung konkreter ComponentManager-Klassen wird eine weitere Restriktionsklasse zur Verfügung gestellt:

ParameterRestrictions

Über diese Klassen können Gültigkeitsintervalle für bestimmte Parameter festgelegt werden. Jede *ParameterRestriction* besitzt folgende Angaben:

- einen Wert, der die unterste Grenze des Gültigkeitsbereichs definiert
- einen Wert, der die oberste Grenze des Gültigkeitsbereichs definiert
- einen Wert, der einen Durchschnittswert für den Parameter definiert
- den augenblicklichenWert des Parameters.

Bei der Spezifikation der Parameterwerte muss auf Beibehaltung der Verhältnismäßigkeiten geachtet werden. Unterschiedliche Parameterangaben einer bestimmten Größe sollten in derselben Maßeinheit spezifiziert werden. Die in dieser Arbeit verwendeten Parameter beschreiben in der Regel Längen, die im Falle der Zellmodellierung in der Maßeineit µm angegeben werden.

Die Klasse BorderManager

Modellierungsrichtlinien, die ausschließlich für Border-Bausteine gelten sollen, werden durch die Klasse *BorderManager* festgelegt. Ein *BorderManager* besitzt alle Elemente eines *ComponentManager* und erweitert diese um folgende Border-Baustein-spezifische Regeln:

- Das Aussehen eines Border-Bausteins wird durch genau ein Shape-Objekt bestimmt. Welche Formen für die Modellierung des Borders verwendet werden dürfen, wird im BorderManager über ShapeRestrictions festgelegt. Jede ShapeRestriction enthält einen String, der die Form benennt, und ParameterRestrictions, die Einschränkungen für die Parameter der Form spezifizieren. Ein Border-Objekt darf nur Shape-Objekte aufnehmen, deren Form in den ShapeRestrictions registriert wurde. Somit ist es möglich zu verhindern, dass z. B. ein zylinderförmiger Zellkern oder ein kegelförmiges Mitochondrium erzeugt wird.
- Das Aussehen eines Border-Objektes ist abhängig vom Aussehen des Border-Objektes der darunterliegenden Schicht. Es muss dafür gesorgt werden, dass ein Shape-Objekt eines logisch niedriger indizierten Borders in der graphischen Darstellung innerhalb des Shape-Objektes des höher indizierten Borders liegt. Um dies zu garantieren, wird die Möglichkeit unterstützt, dass Border-Objekte das Aussehen von ihrem Vorgänger übernehmen und es nicht möglich ist, für diese Objekte eine andere Form zu bestimmen. Soll von dieser Technik Gebrauch gemacht werden, muss ein entsprechendes Flag in der Klasse BorderManager gesetzt werden.
- Wenn ein Baustein nach dem eben beschriebenen Mechanismus das Aussehen seines Vorgängers erbt, muss zusätlich ein Wert für den Abstand zur Vorgängerschicht angegeben werden. Eine entsprechende ParameterRestriction ermöglicht es, ein Gültigkeitsintervall für diesen Abstand zu definieren.

In Abbildung 93 b) werden die Modellierungsrichtlinien für die beiden Membranen eines Zellkerns aufgelistet. Die Zusammenhänge zwischen den einzelnen Managern und den Restriction-Klassen werden in Abbildung 94 dargestellt.

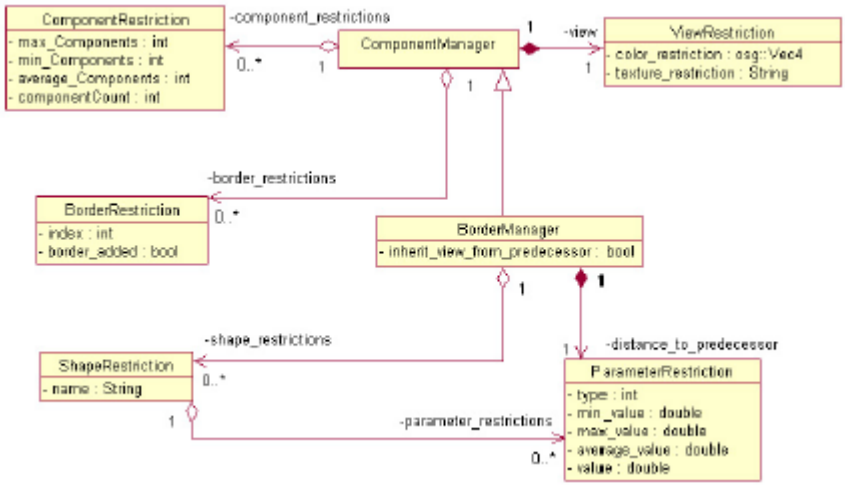


Abbildung 94 Klassendiagramm der Managerklassen

5.2.3.1.5.4 Klassen zur Behandlung unterschiedlicher Zelltypen

Damit der im Kapitel „Zellmodellierung“ dargestellte unterschiedliche Aufbau von Zellen unterschiedlichen Typs bei der Modellierung berücksichtigt werden kann, wurden einige Klassen entwickelt, die nun kurz vorgestellt werden.

Die Klasse ComponentVisitor

Die durch den ComponentManager festgelegten Modellierungsrichtlinien sind abhängig vom zu modellierenden Zelltyp. Für jeden Baustein muss bei Gebrauch ein zelltypabhängiger ComponentManager zur Verfügung gestellt werden.

Eine Möglichkeit würde darin bestehen, dass jede Bausteinklasse das Wissen über die zelltypbedingten ComponentManager-Einstellungen eigenständig verwaltet. Ein bedeutender Nachteil dieser Methode ist allerdings, dass für die Bereitstellung neuer Zelltypen alle Bausteinklassen angepasst werden müssten (siehe Abbildung 95).

Ein weiterer Ansatz besteht darin, für jeden Baustein typabhängige Klassen bereitzustellen, z. B. eine Klasse YeastNucleus, um einen Zellkern einer Hefezelle darzustellen, oder eine Klasse HumanMuscleMitochondrion, um ein Mitochondrium einer menschlichen Muskelzelle darzustellen. Auch bei diesem Ansatz erfordert die Erweiterung modellierbarer Zelltypen einen erheblichen Implementierungsaufwand (siehe Abbildung 96).

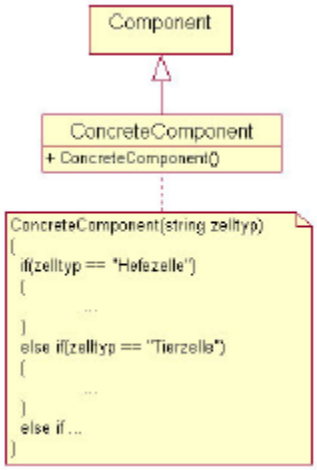


Abbildung 95 Neue Komponente

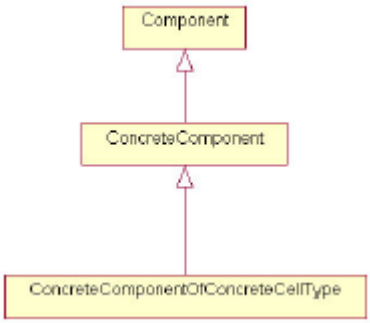


Abbildung 96 Neue Komponente durch Erweiterung

Für das entwickelte Klassenmodell wurde eine dritte Methode entwickelt, die auf dem Visitor-Pattern basiert [165]. In den ersten beiden beschriebenen Ansätzen wurden für jeden Baustein typspezifische Einstellungen definiert; der neue Ansatz sieht vor, für jeden Typ bausteinspezifische Einstellungen zu formulieren. Die Funktionsweise dieser neuen Methode wird nun kurz verdeutlicht:

- Die Component-Klasse stellt eine Accept-Funktion zur Verfügung, über die ein Baustein Visitor-Objekte aufnehmen kann.
- Jede konkrete Visitor-Klasse repräsentiert einen bestimmten Zelltyp und besitzt die kompletten Informationen darüber, wie die unterschiedlichen Bausteine innerhalb eines Objektes des speziellen Zelltyps aufgebaut sind.
- Um einem Baustein einen bestimmten Zelltyp zuzuweisen, wird ein Objekt der entsprechenden Visitor-Klasse zu dem Baustein geschickt, das die benötigten Umstellungen, insbesondere die Einstellungen des ComponentManagers, durchführt.

Die Formulierung von Modellierungsrichtlinien ist allerdings nicht nur abhängig vom Zelltyp, sondern gelegentlich auch vom Submodell, in das der Baustein eingefügt werden soll.

Intrazelluläre Vesikel besitzen unter Umständen einen anderen Aufbau als Vesikel des Golgi-Apparates.

Für Border-Bausteine gilt eine weitere Besonderheit: Die Modellierungsrichtlinien dieser Bausteine müssen in Abhängigkeit von dem durch sie begrenzten Baustein und der Schicht, in der der Border sich befinden soll, formuliert werden. So unterscheidet sich z. B. die innere Membran eines Mitochondriums in ihrer beschreibbaren Struktur deutlich von dessen äußerer Membran.

Alle diese Eigenarten wurden bei der Entwicklung der Klasse ComponentVisitor beachtet, die den Grundaufbau für die typspezifischen Visitor-Klassen darstellt. In Abbildung 97 wird der entwickelte Mechanismus anhand eines Klassendiagramms dargestellt.

Die Klasse Categorisation

Um einen Klassifizierungsbaum für die unterschiedlichen Zelltypen im Programm aufstellen zu können, wurde die Klasse Categorisation eingeführt. Jedes Categorisation-Objekt besitzt einen Namen und eine Liste von Objekten der Klasse Category, die die erste Ebene der Klassifizierungshierarchie darstellen.

Die Klasse Category dient zur Beschreibung von Kategorie-Elementen des Klassifizierungsbaumes. Ein Category-Objekt besitzt folgende Eigenschaften:

- einen Kategorienamen
- eine Liste von Category-Objekten, durch die die Hierarchie des Klassifizierungsbaumes nachgebildet werden kann
- einen Zeiger auf ihr Vorgänger-Category-Objekt
- ein ComponentVisitor-Objekt
- eine Liste weiterer Categorisation-Objekte, welche benötigt wird, um verschiedene Klassifizierungsstufen in einem einzigen Klassifizierungsbaum unterzubringen (siehe Abbildung 98). Bei der Zelle sind dies z. B. die Klassifizierung nach Organismus auf der einen und nach spezialisiertem Zelltyp auf der anderen Seite.

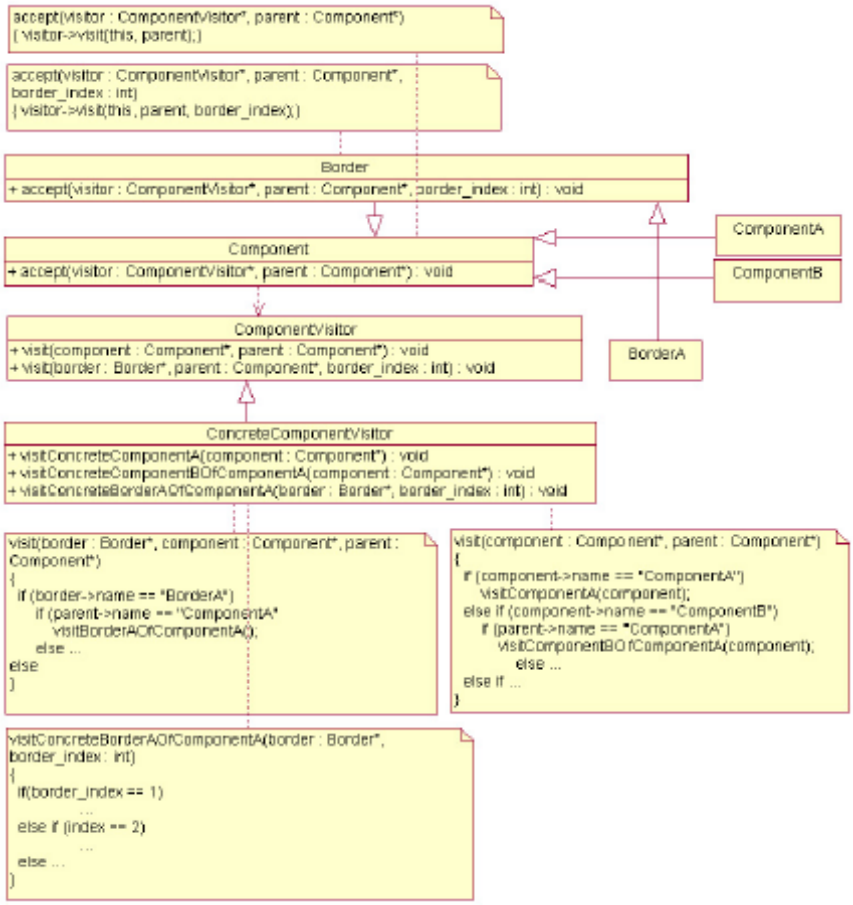


Abbildung 97 Klassendiagramm der Visitor-Klassen

In jedem Categorisation-Objekt kann genau ein Category-Objekt markiert werden. Dadurch kann die Categorisation-Klasse dazu verwendet werden, die Informationen über den Zelltyp zu speichern, der modelliert werden soll. In Abbildung 99 ist ein Klassendiagramm aufgeführt, das den Zusammenhang zwischen der Catgorisation-, der Category- und der ComponentVisitor-Klasse verdeutlicht.

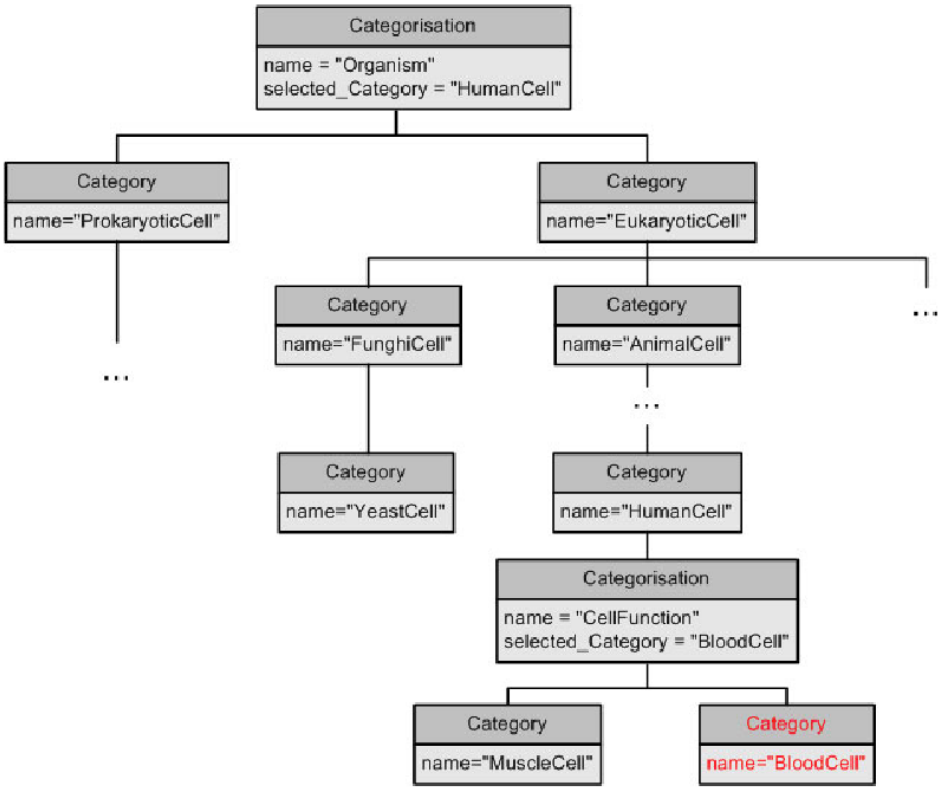


Abbildung 98 Klassifizierungsbaum

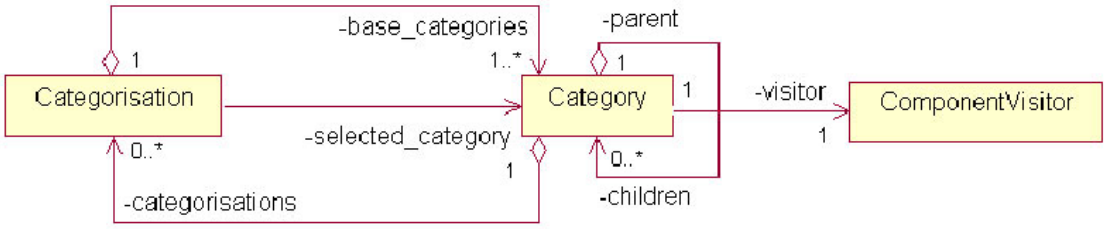


Abbildung 99 Catgorisation-, Category- und ComponentVisitor-Klasse

5.2.3.1.6 Erweiterbarkeit

Die in den vorigen Kapiteln vorgestellten Basisklassen bilden die Grundstruktur zur Entwicklung konkreter Klassen für den Modellierungsprozess. Es werden nun Richtlinien für die Implementierung der Bausteine und der Zelltypen formuliert.

5.2.3.1.6.1 Die Klasse CellLibrary

Alle Bausteine, die Kategorisierung der Zelle und die verschiedenen Visitor-Klassen müssen in einem ComponentLibrary-Objekt registriert werden, um für den Modellierungsprozess zur Verfügung zu stehen. Damit die Registrierung nicht während der Laufzeit erfolgen muss, wird die konkrete ComponentLibrary-Klasse CellLibrary bereitgestellt. Die Klasse ist so ausgerichtet, dass bei der Konstruktion eines Objektes der Klasse die Registrierung der Modellierungselemente automatisch erfolgt. Bei der Erweiterung von Bausteinen oder Zelltypen muss die Klasse CellLibrary entsprechend angepasst werden.

5.2.3.1.6.2 Die Klasse CellVisitor

Die Klasse ComponentVisitor beschreibt die Grundstruktur für die konkreten Visitor-Klassen, die erweiterte Klasse CellVisitor dient als Basis-Visitor für die typspezifischen Visitor-Klassen bei der Zellmodellierung.

5.2.3.1.6.3 Richtlinien zur Erweiterbarkeit

Das hier vorgestellte Klassenmodell stellt eine Grundstruktur von Klassen für die Zellmodellierung zur Verfügung. Diese Grundstruktur muss um konkrete Klassen erweitert werden, um sie für die Zellmodellierung einsetzen zu können. Es werden nun Richtlinien für die Vorgehensweise bei der Neuimplementierung konkreter Modellierungklassen vorgestellt.

Soll ein neuer Baustein im Modellierungskontext bereitgestellt werden, muss eine eigene Basisklasse entwickelt werden. Für diese Klasse sollten folgende Methoden und Attribute definiert werden:

- ein Konstruktor, in dem der Name der Bausteinklasse und der Strukturtyp definiert werden, und gegebenenfalls ein entsprechender Destruktor
- eine Clone-Funktion und ein Copy-Konstruktor
- eine Init-Funktion, falls die Möglichkeit bestehen soll, einen vordefinierten Aufbau des Bausteins durch Angabe verschiedener Parameter automatisch generieren lassen zu können
- spezielle Funktionen und Attribute zur Beschreibung des Bausteins
- eine Serialize- und eine Deserialize-Funktion, falls der Baustein Attribute besitzt, die bei der Serialisierung mitgeschrieben werden müssen.

Gegebenenfalls muss eine neue bausteinspezifische Manager-Klasse entwickelt werden. Damit der Baustein bei der Modellierung verwendet werden kann, muss er zum einen in der CellLibrary hinzugefügt werden, und zum anderen müssen die Visitor-Klassen entsprechend angepasst werden.

Um einen neuen modellierbaren Zelltypen hinzuzufügen, muss zuerst eine neue Visitor-Klasse implementiert werden. Damit diese neuentwickelte Klasse im Zellmodellierungsprozess verwendet werden kann, muss das Categorisation-Objekt aus der CellLibrary um eine neue Kategorie für den neuen Zelltyp erweitert und dieser neuen Kategorie ein Objekt der neuen Visitor-Klasse zugeordnet werden.

5.2.3.1.7 Aufbau eines Zelleditors

Im vorherigen Kapitel wurde ein Konzept zur computerbasierten Darstellung von biologischen Zellen entwickelt; in diesem Kapitel wird nun die erstellte Entwicklungsumgebung zur

Bearbeitung eines Zellmodells, das Programm *CellEdit*, vorgestellt. Zuerst werden Anforderungen an die Funktionalität des Zelleditors zusammengestellt und die Funktionen Selektion und Fokussierung eingeführt. Danach wird die Verwendung des Model-View-Controller-Konzepts für den Zelleditor untersucht und eine Betrachtung der Windowssystem-Abhängigkeiten der Bausteine durchgeführt. Im Anschluss daran werden dann die entwickelten Werkzeuge des Programms vorgestellt und einige Funktionsabläufe beschrieben.

5.2.3.1.7.1 Anforderungen an die Funktionalität des Zelleditors

Es werden nun, ausgehend von der Betrachtung einer als möglich angenommenen Vorgehensweise für die Modellierung einer Zelle mit Hilfe eines entsprechenden Modellierungstools, die Anforderungen an die Funktionalität des Modellierungswerkzeuges durch Auflisten von unterschiedlichen Anwendungsfällen dargestellt.

Der Modellierungsprozess könnte folgendermaßen aussehen:

- Der Benutzer startet ein neues Modellierungsprojekt oder öffnet ein bestehendes Projekt.
- Der Benutzer fügt Bausteine in die Zelle ein und modelliert dadurch den inneren Aufbau der Zelle.
- Die Bausteine werden innerhalb der Zelle durch den Benutzer entsprechend seinen Vorstellungen beliebig positioniert und orientiert.
- Der Benutzer führt Anpassungen an den bausteinspezifischen Eigenschaften durch und erschafft dadurch ein Modell, das neben dem graphischen Aussehen auch in der logischen Struktur seinen Wünschen angepasst ist.
- Einige Bausteine bieten eigenständige Modellierungsfunktionen. Der Benutzer kann in diese Bausteine hineinzoomen und diese entsprechend ihren bereitgestellten Modellierungsmethoden individuell bearbeiten.
- Hat der Benutzer den Modellierungsprozess fürs Erste abgeschlossen, speichert er das erstellte Projekt ab.

5.2.3.1.7.2 Anwendungsfälle

Ausgehend von dem erdachten Modellierungsprozess werden nun Anwendungsfälle aufgelistet, deren Durchführung über die implementierte graphische Benutzerschnittstelle ermöglicht werden muss. In Abbildung 100 ist ein für den Zelleditor entworfenes UML-UseCase-Diagramm dargestellt; die darin aufgezeigten Anwendungsfälle werden nun näher beschrieben:

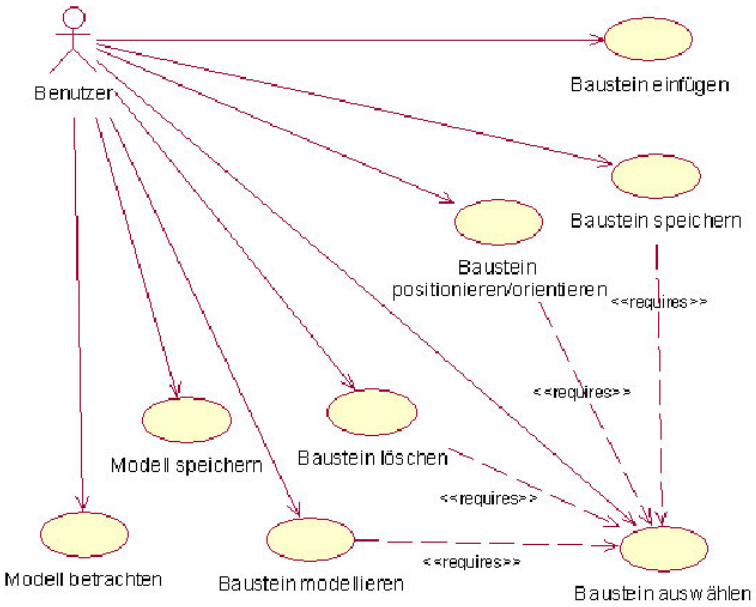


Abbildung 100 UML-UseCase-Diagramm zur Beschreibung der Anwendungsfälle des Zellmodellierungsprozesses

Baustein einfügen

Dieser Anwendungsfall beschreibt das Einfügen eines neuen Komponentenbausteins in das Zellmodell. Hierbei sind drei unterschiedliche Einfügeoperationen denkbar:

- Ein Baustein wird neu erstellt und in das Modell eingefügt.
- Ein auf einem Speichermedium abgelegter Baustein wird in das Programm eingeladen und in das Modell eingefügt.
- Ein Baustein wird in einer Zwischenablage gehalten und von dort aus in das Modell eingefügt.

Baustein auswählen

Beabsichtigt der Benutzer, Veränderungen an einem bestimmten Baustein durchzuführen, muss er dem Programm zuerst mitteilen, welchen Baustein er zu verändern beabsichtigt. Der Anwendungsfall ”Baustein auswählen“ beschreibt diesen Auswahlvorgang. Für die Zellmodellierung werden zwei Auswahlmodi eingeführt, die nun kurz erläutert werden:

Selektion

Die Auswahl eines Bausteins zum Durchführen von Manipulationsoperationen wird als Selektion bezeichnet. Die Selektion eines Bausteins ist Grundvoraussetzung für die Anwendungsfälle ”Baustein löschen“, ”Baustein speichern“, ”Baustein positionieren/orientieren“ und ”Bausteineigenschaften ändern“.

Fokussierung

Die Anwahl eines Bausteins als neue Modellierungsebene, das Hineinzoomen in einen Baustein, wird als Fokussierung bezeichnet und ist Voraussetzung des Anwendungsfalls ”Baustein modellieren“.

Baustein löschen

Bausteine, die in das Modell eingefügt wurden, müssen auch wieder entfernt werden können. Auch hier sind mehrere Fälle zu unterscheiden:

- Ein Baustein wird aus dem Modell entfernt und im Programm komplett gelöscht.
- Ein Baustein wird aus dem Modell entfernt, aber in einer Zwischenablage gepeichert.

Baustein speichern

Für den Anwendungsfall, der das Abspeichern eines Bausteins beschreibt, sind zwei unterschiedliche Speicherszenarien denkbar:

- Ein Baustein wird in eine Datei auf einem Speichermedium geschrieben.
- Ein Baustein wird in einer Zwischenablage gespeichert.

Baustein positionieren/orientieren

Dieser Anwendungsfall beinhaltet alle Aktionen, durch die ein Baustein im Modell transformiert, d. h. im Besonderen rotiert oder positioniert werden kann.

Bausteineigenschaften verändern

Ein Baustein besitzt eventuell spezifische Eigenschaften, die vom Benutzer geändert werden können. Die Aktionen zur Durchführung dieser Änderungsoperationen werden durch den Anwendungsfall ”Bausteineigenschaften verändern“ beschrieben.

Baustein modellieren

Ein Baustein kann einen eigenen Modellierungskontext bilden und als eigenständiges Modell bearbeitet werden. Alle Operationen, die zu diesem Zweck durchgeführt werden, sind in dem Anwendungsfall ”Baustein modellieren“ zusammengefasst.

Modell speichern

Dieser Anwendungsfall umfasst alle Aktionen, die zum Abspeichern der Modelldaten des gesamten Projektes durchgeführt werden.

Modell betrachten

Der Benutzer kann sich das erstellte Modell in seinem graphischen oder logischen Aufbau ansehen. Der Anwendungsfall ”Modell betrachten“ bezieht sich auf alle Aktionen, durch die sich der Benutzer einen Überblick über das erstellte Modell verschaffen kann.

Bei der Entwicklung des Zelleditors wurden die aufgestellten Anforderungen berücksichtigt. Die Begriffe Selektion und Fokussierung, die im Zusammenhang mit den in diesem Abschnitt durchgeführten Betrachtungen eingeführten wurden, sollen nun näher erläutert werden.

5.2.3.1.7.3 Selektion und Fokussierung

Die Methode der zwei im vorigen Abschnitt vorgestellten Auswahlmodi *Selektion* und *Fokussierung* wird in diesem Abschnitt kurz dargestellt.

Selektion

Selektierte Elemente befinden sich immer unmittelbar innerhalb des Modellierungskontextes des fokussierten Elementes. Besitzt z. B. das Wurzelelement der Zelle den Fokus, dann kann der Zellkern, nicht aber das Kernkörperchen, das ein unmittelbares Element des Zellkerns und nicht der Zelle ist, selektiert werden. Ein selektiertes Element soll im graphischen Modell transformiert, gelöscht, gespeichert und an den Eigenschaften verändert werden können.

Es ist immer genau ein Element selektiert, zumindest das fokussierte Element selbst. Besitzt der Fokus gleichzeitig auch die Selektion, muss darauf geachtet werden, dass das fokussierte Element nicht in der graphischen Darstellung transformiert werden kann, da es selbst das Wurzelelement des aktuellen Modells bildet und das absolute Koordinatensystem der 3-D-Szene definiert.

Fokussierung

Als Fokussierung wird der Übergang in einen neuen Modellierungskontext bezeichnet. Beim Starten eines neuen Modellierungsprojektes befindet sich der Fokus zuerst auf dem Wurzelelement der Zelle; für die Modellierung anderer Bausteine, z. B. um dem Zellkern der Zelle ein Kernkörperchen hinzuzufügen, muss der Fokus zuerst auf den neuen zu modellierenden Baustein wechseln.

5.2.3.1.7.4 Model-View-Controller

Nachdem die konzeptionellen Grundlagen über die Funktionalität des Editors bereits aufgeführt wurden, sollen jetzt weitere Überlegungen über den Aufbau der Werkzeuge des Modellierungstools dargelegt werden. Alle Werkzeuge des Editors greifen auf ein einziges zu bearbeitendes Zellmodell zu. Jedes dieser Werkzeuge kann das Modell auf bestimmte Art und Weise manipulieren und ist in Hinblick auf den eigenen Aufbau und die eigene Funktionalität abhängig vom Zustand des Zellmodells. Veränderungen der Daten des Modells, aber auch Selektionen oder Fokussierungen von Bausteinen durch ein bestimmtes Werkzeug bedingen eventuell die Anpassung anderer Werkzeuge an die veränderte Modellumgebung.

Um diese gegenseitige Beeinflussung managen zu können, wurde bei der Implementierung des Editors auf das Model-View-Controller-Konzept [165] zurückgegriffen. Das entwickelte Klassenkonzept für die drei Elemente Model, View und Controller wird nun beschrieben (siehe dazu Abbildung 101).

Das Model – Die Klasse VirtualWorld

Die Klasse VirtualWorld repräsentiert in der entwickelten Modellierungsumgebung das „Model“ des MVC-Konzeptes. Bei einem laufenden Modellierungsprojekt ist immer genau ein VirtualWorld-Objekt geöffnet, das folgende Projektdaten verwaltet:

- Das VirtualWorld-Objekt enthält den Wurzel-Baustein des Projekts und darüber alle Informationen über den kompletten Aufbau des Zellmodells.
- Innerhalb der Baustein-Hierarchie des Modells kann ein beliebiger Baustein selektiert werden. Im VirtualWorld-Objekt wird das Wissen über den aktuell selektierten Baustein gespeichert.

- Die Information darüber, welcher Baustein den momentanen Fokus besitzt und als Submodell bearbeitet wird, ist ebenfalls in dem VirtualWorld-Objekt vorhanden.
- Ein VirtualWorld-Objekt kann einen beliebigen Baustein zwischenspeichern und erfüllt dadurch die Funktion einer Zwischenablage. Zur Realisierung der Benachrichtigung der Werkzeuge bei Veränderung der Modell-Daten wird das Observer-Pattern verwendet. Die Klasse VirtualWorld verwaltet eine Liste von Observer-Objekten. Bei einer Veränderung der Modell-Daten werden alle registrierten Observer vom VirtualWorld-Objekt über die Änderung informiert, wodurch ihnen die Möglichkeit geboten wird, entsprechend auf diese Änderungen zu reagieren.

Damit dieser Mechanismus der Nachrichtenübermittlung funktioniert, muss die Änderung der Daten über die von der VirtualWorld-Klasse zur Verfügung gestellten Schnittstellen realisiert werden. Die Manipulationsfunktionen, die durch die Klasse VirtualWorld abgedeckt sind, werden nun aufgelistet und kurz erläutert:

Baustein hinzufügen

Ein Baustein wird an das VirtualWorld-Objekt übergeben und von diesem Objekt in die Szene des augenblicklich fokussierten Elementes eingefügt.

Baustein entfernen

Dem VirtualWorld-Objekt wird ein Baustein genannt, der von dem Objekt aus dem Modell entfernt wird.

Baustein selektieren

Bei der Selektion eines Bausteins gelten folgende Regeln, die sich aus den im vorherigen Abschnitt dargestellten Selektions-Eigenschaften ergeben:

- Befindet sich das ausgewählte Element unmittelbar innerhalb des fokussierten Objektes oder ist das Element selbst im Besitz des Fokus, wird im Modell das gewählte Objekt selektiert.
- Befindet sich das ausgewählte Element außerhalb der fokussierten Komponente, d. h., das Element ist kein Unterbaustein der untersten Modellierungsebene des fokussierten Bausteins, dann wechselt der Fokus zu der Elternkomponente des gewählten Elements, und das gewählte Element selbst wird selektiert.
- Wird das unterste Element des Baumes, das Wurzelelement angewählt, so wechseln Fokus und Selektion zu diesem Element.
- Für Bausteine der komplexen und der begrenzenden Strukturform gelten besondere Regeln. Da solche Bausteine im Modell nicht bewegt werden sollten, werden Bausteine der eben genannten Strukturtypen bei der Selektion gleichzeitig fokussiert.

Baustein fokussieren

Mit dem Fokus wechselt auch die Selektion zu einem neuen Baustein.

Zwischenablagefunktionen

Der Zugriff auf die Zwischenablage des VirtualWorld-Objekts erfolgt über die üblichen Funktionen.

Baustein kopieren

Ein an das VirtualWorld-Objekt übergebener Baustein wird geklont, und das geklonte Element wird im VirtualWorld-Objekt zwischengespeichert.

Baustein ausschneiden

Ein übergebener Baustein wird geklont und aus dem Modell entfernt. Anschließend wird das geklonte Element im VirtualWorld-Objekt zwischengespeichert.

Baustein einfügen

Der Baustein, der im VirtualWorld-Objekt zwischengespeichert ist, wird in die Szene des fokussierten Elements eingefügt.

Von der Klasse VirtualWorld werden keine Schnittstellen bereitgestellt, um Positionierungs- bzw. Orientierungsoperationen oder Änderungsoperationen an bausteinspezifischen Eigenschaften durchzuführen. Solche Operationen werden direkt an den betroffenen Bausteinen ausgeführt und erzwingen folglich keine Benachrichtigung der Observer-Objekte. Durch Aufruf der Notify-Methode des VirtualWorld-Objektes kann eine solche Benachrichtigung allerdings manuell erzwungen werden.

5.2.3.1.7.5 Die Views

Alle Werkzeuge, die die Observer-Schnittstelle realisieren und dem VirtualWorld-Objekt als Empfänger von Änderungsnachrichten gemeldet sind, sind die „Views“ im Sinne des MVC-Konzeptes. Zu den Views der MVC-Umgebung des Zelleditors zählen neben dem Hauptfenster noch der Szenenbetrachter, die Baustein-Buttonleiste, das Eigenschafts-Bedienelement der Bausteine und der Hierarchie-Browser. Diese Werkzeuge werden in Abschnitt 5.5 eingehend beschrieben.

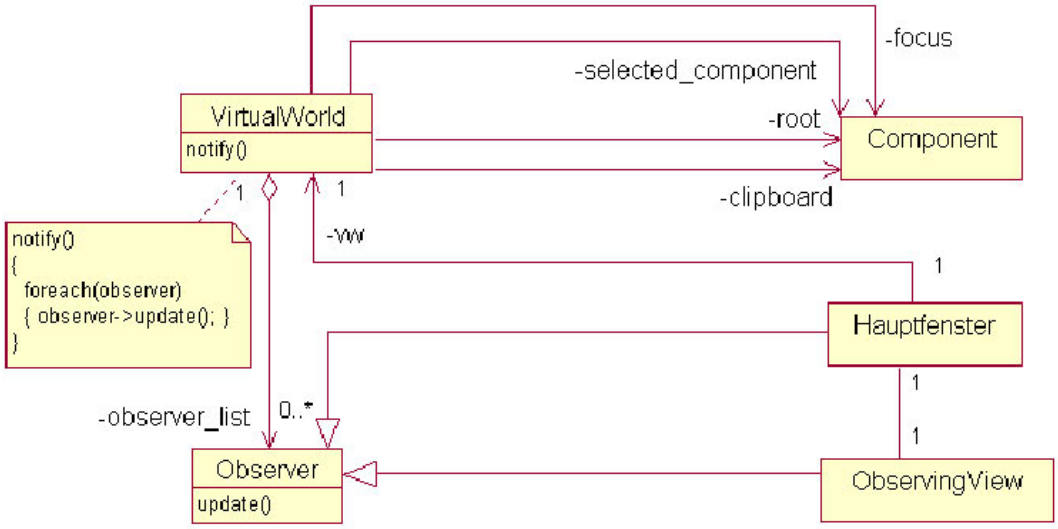


Abbildung 101 Das Model-View-Controller-Konzept des Zelleditors

5.2.3.1.7.6 Der Controller

Das Hauptfenster ist im MVC-Konzept des Editors „View“ und „Controller“ zugleich. Die GUI-Elemente des hier verwendeten FOX-Toolkits [166] sind so ausgerichtet, dass sie ihre

Events selbst managen können. Der Controller muss also nicht zwangsläufig das Event-Management der einzelnen Observer-Objekte übernehmen.

Für den Controller ist ein anderer Aufgabenbereich vorgesehen: Die einzelnen Views führen die Änderungen an dem VirtualWorld-Objekt nicht eigenständig durch, sondern senden eine Nachricht an das Hauptfenster, in der sie diesem die Änderungswünsche mitteilen. Das Hauptfenster wertet diese Nachrichten aus, führt die Änderungen an den Daten des VirtualWorld-Objektes aus und besitzt schließlich noch die Möglichkeit, weitere senderabhängige Änderungen durchzuführen, die nicht über die VirtualWorld-Klasse geregelt werden. Dieser Mechanismus ist in Abbildung 102 dargestellt.

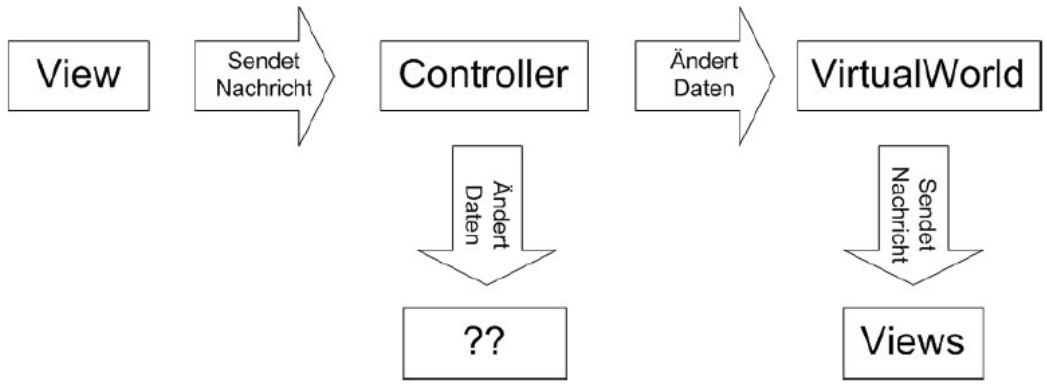


Abbildung 102 Nachrichtenkette

5.2.3.1.8 Vorstellung der Benutzeroberfläche

In diesem Abschnitt werden die einzelnen Elemente der entwickelten Benutzeroberfläche des Zellmodellierungswerkzeuges vorgestellt. Dazu wird zunächst der Aufbau des Hauptfensters erläutert.

5.2.3.1.8.1 Hauptfenster

Das Hauptfenster des Zelleditors wird durch die Klasse FXCellEditMainFrame beschrieben. Alle Werkzeuge des Zelleditors sind in diesem Hauptfenster angeordnet. Wenn noch kein Modellierungsprojekt geöffnet wurde, besitzt das Hauptfenster nur eine Menü-, eine Werkzeug- und eine Statusleiste. Nach Öffnen eines Projektes werden ein Szenenbetrachter, eine Baustein-Buttonleiste, ein Karteikarten-Bedienelement und ein Hierarchie-Browser in das Hauptfenster eingefügt (siehe Abbildung 103).

Das Hauptfenster dient als Controller für die verschiedenen Views und verwaltet diejenigen Werkzeuge, die nicht als View implementiert sind und somit nicht über Änderungen des Zellmodells informiert werden. Zu diesen Werkzeugen zählen die Menü- und dieWerkzeugleiste, die Statusleiste und das Karteikarten-Bedienelement.

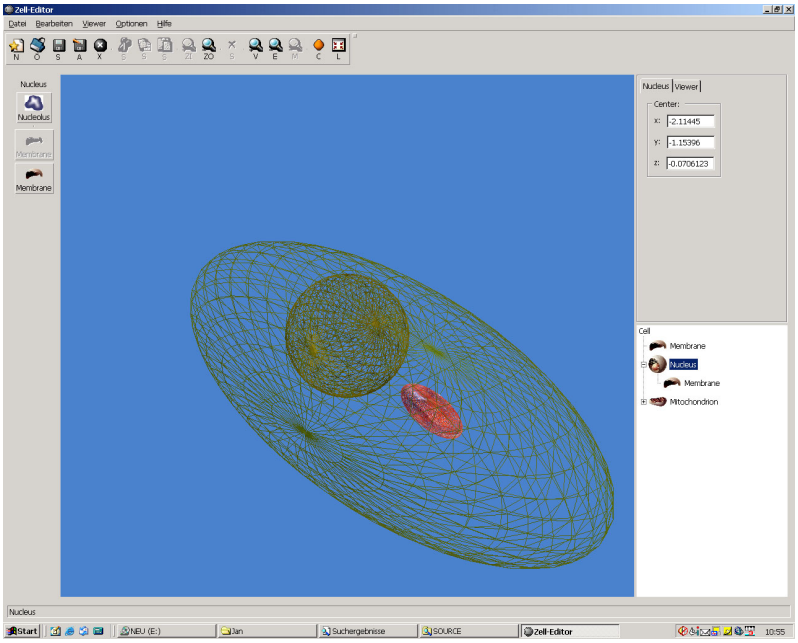


Abbildung 103 Das Hauptfenster des Zelleditors

Weiterhin besitzt das Hauptfenster Objekte der Klasse CellLibrary und der Klasse FXCellLibrary, die die Modellierungsinformationen für eine Zelle beinhalten, und ein VirtualWorld-Objekt, das die Datenstruktur des Modellierungsprojektes verwaltet.

Die vom Hauptfenster verwalteten Elemente werden nun kurz beschrieben.

Menü- und Werkzeugleiste

Das Hauptfenster enthält eine bei graphischen Benutzerschnittstellen übliche Menü- und Werkzeugleiste, über die bestimmte Operationen durchgeführt werden können. Der Aufbau dieser Leisten kann von anderen Werkzeugen erweitert werden.

Statusleiste

In der Statusleiste am unteren Ende des Zelleditors können bestimmte Hinweise für den Benutzer übermittelt werden. Wird beispielsweise der Mauszeiger auf ein bestimmtes GUI-Element bewegt, können in der Statusleiste Informationen über das entsprechende Element ausgegeben werden.

Karteikartenelement

Das Karteikartenelement ermöglicht es, mehrere Einstellungsbedienfelder auf einer kleinen Fläche zu vereinen. Bislang besitzt das Karteikartenbedienfeld zwei Karteikarten: eine für das Bedienfeld für bausteinspezifische Eigenschaften und eine für Einstellungen an dem im nächsten Kapitel vorgestellten Szenenbetrachter.

5.2.3.1.8.2 Der Szenenbetrachter

Um das 3-D-Modell der Zelle in seinem graphischen Aussehen bearbeiten zu können, muss ein Werkzeug zur Verfügung stehen, das das Modell oder Teilbereiche das Modells auf dem

Bildschirm darstellt und Funktionen anbietet, um die 3-D-Szene zu erkunden und 3-D-Objekte innerhalb der Szene positionieren oder orientieren zu können.

In kommerziellen 3-D-Modellierungstools, wie z. B. 3-D-Studio Max [167], wird die Szene im Normalfall in vier verschiedenen Ansichten, sogenannten Viewports, dargestellt. Drei dieser Viewports zeigen Orthogonalprojektionen der Szene auf die drei durch die Achsen des Koordinatensystems definierten Ebenen, die als Links-, Oben- und Vorne-Ansicht bezeichnet werden (Seitenansicht, Aufsicht, Frontansicht). Ein weiterer Viewport bietet eine perspektivische Darstellung der Szene, ausgehend von einer bestimmten definierten Kamera.

Diese Bereitstellung verschiedener Ansichten ist notwendig, um die Übersichtlichkeit bei der Modellierung von geometrischen Daten, z. B. das Transformieren von Punkten, Linien oder einzelnen Flächen, zu garantieren. Die Zellmodellierung erfolgt allerdings im Wesentlichen durch Verschieben und Drehen der einzelnen Bausteine.

Solche Operationen lassen sich am intuitivsten innerhalb eines Viewports mit perspektivisch projizierter Szene steuern, da durch die perspektivische Abbildung die Tiefeninformationen der Szene übermittelt werden und sich der Benutzer somit innerhalb der Szene räumlich orientieren kann.

Für den Zelleditor wird nur ein einziger, im Programm als Szenenbetrachter bezeichneter Viewport zur Verfügung gestellt. Der Szenenbetrachter umfasst ein Rendering-Fenster mit spezieller Ereignisbehandlung für Maus- und Tastaturinteraktionen und einige weitere Tools und Bedienelemente zum Ändern von Darstellungseigenschaften.

Der Szenenbetrachter setzt sich aus mehreren Klassen zusammen. Die Klasse Virtual-WorldViewer definiert einen GUI-unabhängigen Betrachter, die Klasse FXVirtual-WorldViewer ist die Erweiterung dieser Klasse für die FOX-Umgebung. Die Klasse Viewer beschreibt einen virtuellen Betrachter, und die Klasse EventHandler wird zur Ereignisverwaltung eingesetzt. Diese Klassen werden im Folgenden ausführlich beschrieben.

Die Klasse Viewer – ein virtueller Betrachter

Für den Szenenbetrachter wird ein virtueller Betrachter beschrieben, der sich innerhalb der 3-D-Welt befindet. Dieser Betrachter kann durch drei Parameter eindeutig definiert werden:

- einen Positionsvektor, der den Aufenthaltsort des Betrachters angibt (Eye-Position)
- einen Vektor, der die Blickrichtung des Betrachters definiert (Viewing-Direction)
- einen Vektor, der die räumliche Orientierung des Betrachters spezifiziert (Up-Direction).

Der Up-Vektor kann beliebig in den Raum zeigen, es wird allerdings nur die Projektion dieses Vektors auf die senkrecht zur Blickrichtung liegende Ebene berücksichtigt.

Ausgehend von den Informationen über die Position und Ausrichtung des virtuellen Betrachters wird die View-Matrix berechnet, durch die die Weltkoordinaten so transformiert werden, dass der Ursprung in den Augpunkt des Viewers übergeht, die positive y-Achse in Richtung des Orientierungsvektors und die positive z-Achse in Richtung des Blickrichtungsvektors zeigt.

Der virtuelle Betrachter kann innerhalb der Szene beliebig gedreht oder verschoben werden, um die Szene von allen möglichen Blickwinkeln aus betrachten zu können. Translationsbewegungen verändern den Augpunktvektor, Drehbewegungen um den Augpunkt wirken sich auf die zwei Richtungsvektoren aus, Drehbewegungen um einen beliebigen Punkt im Raum verändern alle drei Vektoren.

Die Klasse Viewer umfasst alle Eigenschaften und Methoden des virtuellen Betrachters. Über diese Klasse werden nicht nur Position und Ausrichtung des Betrachters und damit die View-Matrix der Koordinatentransformations-Pipeline definiert, in diese Klasse werden zusätzlich Eigenschaften und Funktionen zum Spezifizieren der Projektionsmatrix hinzugefügt.

Um die Szene über eine perspektivische Projektion auf die Bildfläche abzubilden, wird auf die OpenGL-Funktion gluPerspective zurückgegriffen. Über diese Funktion wird eine perspektivische Projektion definiert, deren Projektionszentrum im Ursprung des View-Koordinatensystems liegt (Augpunkt) und dessen Projektionsfläche parallel zur x-y-Ebene in Blickrichtung des Betrachters aufgespannt ist.

Durch die Near- und Far-Clipping-Ebenen wird ein Pyramidenstumpf definiert (siehe Abbildung 104), der für Clipping-Operationen eingesetzt werden kann. Es werden hierbei nur Objekte gezeichnet, die sich innerhalb des Pyramidenstumpfes befinden.

Die Clipping-Operationen werden für den Szenenbetrachter allerdings ausgeschaltet, damit es möglich ist, sich auch nahe an Objekte heranzubewegen. Das Breite-Höhe-Verhältnis sollte dem Seitenverhältnis des Rendering-Fensters entsprechen.

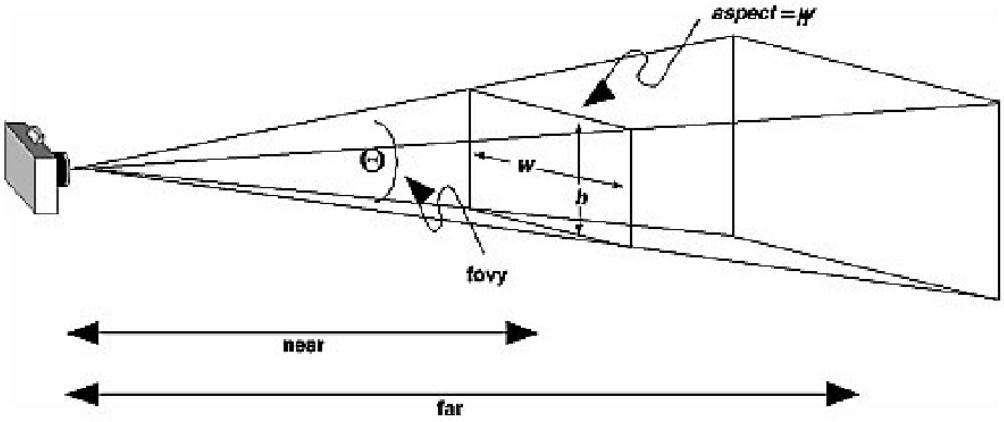


Abbildung 104 Der Pyramidenstumpf einer perspektivischen Projektion (gluPerspective)[WoND99]

EventHandler

Zur Manipulation von Objekten der Szene und zur Manipulation des virtuellen Betrachters wurde die Klasse EventHandler konzipiert. Ein EventHandler-Objekt empfängt Maus- und Tastaturinteraktionen des Benutzers und führt entsprechende Manipulationsoperationen durch.

Es sind insgesamt vier konkrete EventHandler-Klassen entwickelt worden. Zwei dieser Klassen, die Klasse SceneExaminer und die Klasse SceneExplorer, werden für Viewer-Manipulationen, die anderen zwei, die Klasse ObjektManipulator und die Klasse ObjectInserter, für Objekt-Manipulationen verwendet. Die bereitgestellten EventHandler werden nun kurz vorgestellt.

SceneExaminer

Der SceneExaminer-EventHandler wird eingesetzt, wennn ein Objekt der 3-D-Szene betrachtet werden soll. Die Maussteuerung dieses EventHandlers ist so ausgerichtet, dass dem Benutzer

das Gefühl vermittelt wird, er würde das Betrachtungsobjekt in der Hand halten und es durch Drehen und Verschieben der Hand betrachten.

Durch Drücken der linken Maustaste und zeitgleiches Bewegen der Maus kann eine Drehung durchgeführt werden. Es wird allerdings nicht das Objekt selbst gedreht, da durch diesen EventHandler keine Objektmanipulation erfolgen soll, sondern der virtuelle Betrachter wird in Gegenrichtung zur erwünschten Objektdrehung um den Objektmittelpunkt rotiert. Das Ziehen der Maus bei gedrückter rechter Maustaste verschiebt den virtuellen Betrachter auf einer zur Projektionsfläche paralellen Ebene, die den Mittelpunkt des Betrachtungsobjektes besitzt. Die Translationsparameter werden dadurch so berechnet, dass der Mauszeiger stets auf dem Objekt verweilt.

Drehen des Mausrades bzw. Drag-Operationen bei gedrückter mittlerer Maustaste verschieben den virtuellen Betrachter auf der vom Mittelpunkt des Viewports senkrecht in die Szene zeigenden Geraden. Dadurch kann man sich näher an das Objekt heranbewegen oder sich von ihm entfernen. Durch Klicken mit der linken Maustaste auf ein bestimmtes Objekt in der Szene kann dieses Objekt selektiert werden.

Dieser EventHandler ist nur geeignet, wenn sich der virtuelle Betrachter außerhalb und in einiger Entfernung zum Betrachtungsobjekt aufhält. Andernfalls sollte auf den SceneExplorer-EventHandler zurückgegriffen werden.

SceneExplorer

Der SceneExplorer-EventHandler wird verwendet, wenn der Benutzer nicht nur ein bestimmtes Objekt von außen betrachten, sondern auch das Innenleben der Objektszene erkunden will. Durch die für diesen EventHandler bereitgestellte Maussteuerung bekommt der Benutzer das Gefühl, als befände er sich selbst in der Szene – er übernimmt die Rolle des virtuellen Betachters.

Mausbewegungen bei gedrückter linker Maustaste werden in Drehungsoperationen des virtuellen Betrachters um den Augpunkt berechnet. Verschieben der Maus bei gedrückter rechter Maustaste verschiebt den virtuellen Betrachter parallel zur Projektionsfläche in seitlicher Richtung des Viewports.

Die Bewegungen, die über das Mausrad oder die mittlere Maustaste durchgeführt werden können, entsprechen den Bewegungen des SceneExaminers. Auch bei diesem EventHandler kann ein bestimmtes Objekt durch Klicken mit der linken Maustaste selektiert werden.

ObjectManipulator

Dieser EventHandler wird für Transformationen von Objekten in der 3-D-Szene eingesetzt. Die Maussteuerung entspricht der Steuerung des SceneExaminers, jedoch mit dem Unterschied, dass nicht der virtuelle Betrachter, sondern das selektierte Objekt entsprechend bewegt werden.

Durch Mausbewegungen bei gedrückter linker Maustaste werden Drehoperationen des Objektes um dessen Mittelpunkt durchgeführt. Durch Verschieben der Maus bei gedrückter rechter Maustaste wird das Objekt auf einer zur Projektionsfläche paralellen Ebene, die den Mittelpunkt des Betrachtungsobjektes besitzt, verschoben.

Durch Mausrad-Bewegungen oder Verschieben der Maus mit gedrückter mittlere Maustaste kann das Objekt auf einer zur Projektionsfläche senkrecht stehenden Geraden, die durch den Mittelpunkt des Objektes verläuft, bewegt werden.

ObjectInserter

Wenn ein neues Objekt in das Modell eingefügt werden soll, wird der ObjectInserter-EventHandler aktiviert. Bei diesem Handler wird bei Verschieben der Maus ohne Betätigen einer Taste das Objekt wie bei einer Mausbewegung mit gedrückter linker Maustaste innerhalb des ObjectManipulators bewegt. Mausrad- oder Mausbewegungen bei gedrückter mittlerer Maustaste entsprechen den Bewegungen des ObjectManipulators. Durch Klicken mit der linken Maustaste kann die Einfügeoperation abgeschlossen werden.

Die Klasse VirtualWorldViewer

Es wurde zunächst eine Klasse VirtualWorldViewer entwickelt, die einen GUI-unabhängigen Viewport mit Interaktionsmöglichkeiten für eine durch ein VirtualWorld-Objekt repräsentierte 3-D-Szene beschreibt.

In der Klasse VirtualworldViewer wird die Basis des zu rendernden Szenengraphen gehalten. Innerhalb dieser Klasse werden auch die benötigten Matrizen für die View-, Projection- und Viewport-Koordinatentransformationen berechnet und gespeichert.

Die Klasse VirtualWorldViewer enthält außerdem ein Viewer-Objekt und einen Event-Handler, der beliebig ausgetauscht werden kann.

Weiterhin werden einige weitere wichtige Funktionen zur Verfügung gestellt:

Baustein anschauen

Mit Hilfe diese Funktion wird der Betrachter so gedreht, dass der Mittelpunkt des Viewports in den Mittelpunkt des Bausteins, der betrachtet werden soll, übergeht. Der neue Blickrichtungsvektor läßt sich aus der Differenz von Baustein-Mittelpunkt und Augpunkt des Betrachters bestimmen.

Baustein-Zentrum aufsuchen

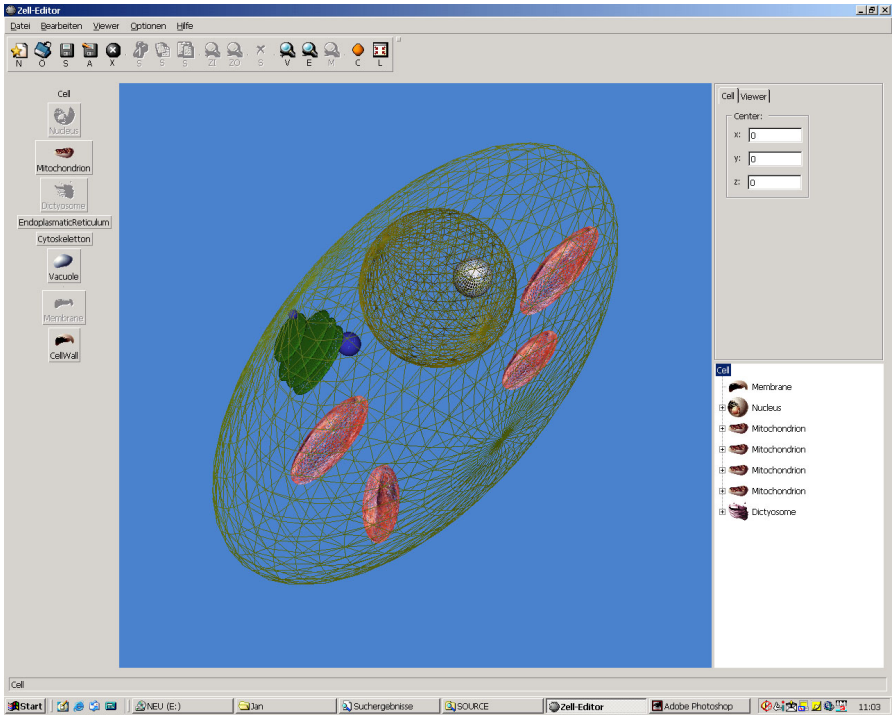
Durch diese Funktion begibt sich der Betrachter in den Mittelpunkt eines Bausteins. Als neuer Augpunkt-Vektor des Betrachters wird dafür der Mittelpunkt-Vektor des Bausteins ausgewiesen.

Bausteinansicht zentrieren

Die Funktion ”Baustein ansehen“ richtet den Blick des Betrachters auf einen bestimmten Baustein aus. Die Funktion ”Bausteinansicht zentrieren“ bewegt den Betrachter nach der Änderung der Blickrichtung zusätzlich so auf der durch die neue Blickrichtung gegebenen senkrecht zur Projektionsebene stehenden Geraden, dass der zu zentrierende Baustein den kompletten Viewport ausfüllt. Für den Baustein wird hierfür zunächst eine Bounding-Sphere berechnet, und anschließend wird der Betrachter so bewegt, dass er sich in dem durch den Radius der Bounding-Sphere gegebenen Abstand vom Mittelpunkt der Bounding-Sphere des Bausteins befindet.

Baustein in den Viewport bewegen

Durch diese Funktion wird der selektierte Baustein so in der Szene platziert, dass er den kompletten Viewport ausfüllt. Diese Funktion ist nützlich, wenn z. B. ein neu in die Szene eingefügter Baustein in den Sichtbereich des Betrachters bewegt werden soll.



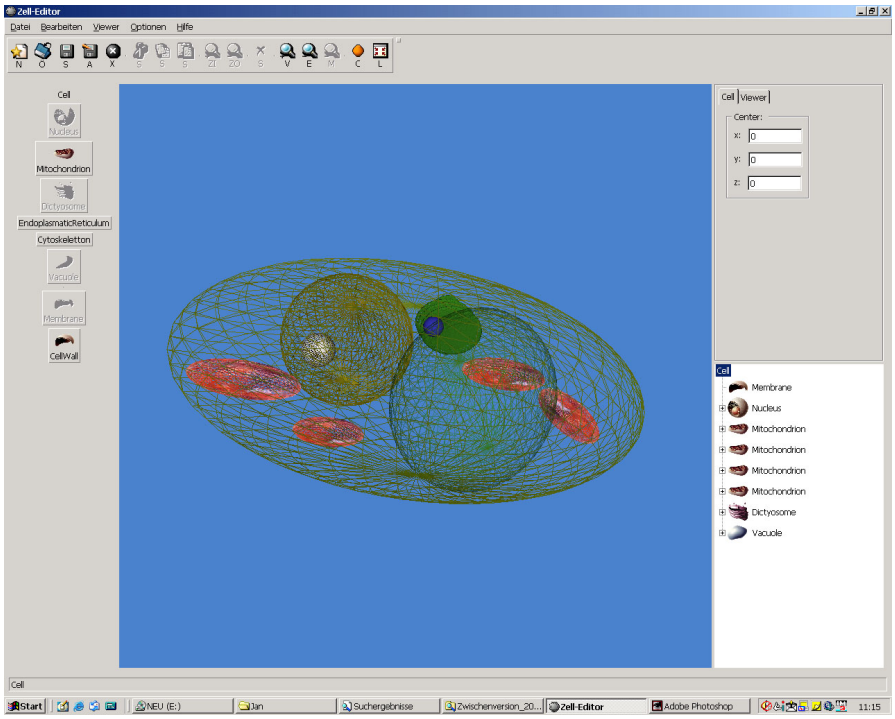


Abbildung 105 Screenshots des Zelleditors

5.2.3.2 Der Simulationskonfigurator

Können im Zelleditor Zellen interaktiv modelliert werden, so ist dies der Beginn eines Simulationsexperiments. Obwohl es keine grundsätzliche Beschränkung gibt, ist der Zelleditor zur Modellierung komplexer Entitäten oberhalb des molekularen Maßstabs gedacht, also für Objekte wie Organellen, Membranen etc. Zur Vervollständigung einer Konfiguration eines Simulationsexperiments bedarf es nun noch der Definition der simulationsrelevanten Moleküle (je nach System bezeichnet als Species, Compound etc.). Wie im Kapitel zur Anforderungsanalyse bereits beschrieben wurde, bedeutet „Definition“ lediglich die Angabe der Stoffmengen, allenfalls unter Angabe des Kompartimentes, in welchem der Stoff vorhanden sein soll. (Ein Kompartiment ist beispielsweise „Cytoplasma“.) Dieser Konfigurator erweitert diese Funktionalität durch eine genaue Positionierbarkeit der Stoffe.

Entsprechend den Erfordernissen einer interaktiven 3-D-Anwendung hält der Konfigurator einige allgemeine Funktionalitäten zur Steuerung und Navigation in der Szene vor:

- Steuergeschwindigkeit ändern
- Steuerung invertieren
- Reset der Szene
- Refresh der Szene
- Hilfe einblenden/ausblenden
- GUI einblenden/ausblenden.

Funktionalitäten wie *Steuergeschwindigkeit ändern* oder *Steuerung invertieren* stehen im Zusammenhang mit der Nutzung von „konventionellen“, sprich 2-D-Eingabegeräten.

5.2.3.2.1 Einfügen von Stoffen

Kernstück des Konfigurators sind Funktionalitäten im Kontext des Einfügens und Entfernens von Stoffen in die Szene. Im Einklang mit dem „Hands-on“-Ansatz, wie er für die *Virtual Glove Box* vorgestellt wurde, wurde auch hier nach Metaphern gesucht, die einen intuitiven Zugang zu der Anwendung ermöglichen.

Es wurden drei wesentliche Funktionen identifiziert:

- freies Einbringen von Stoffen im Raum
- Aufbringen bzw. Einbringen von Stoffen auf der Oberfläche anderer Entitäten, insbesondere auf/in Membranen
- Löschen von eingebrachten Stoffen.

Während für die Löschung eines Stoffes lediglich ein Auswählen per Berührung bzw. Mausklick nötig war, wurde für die beiden Einfügeoperationen jeweils eine eigene Metapher gewählt.

Für das freie Positionieren von Stoffen wird die Metapher einer Injektionsspritze verwendet (siehe Abbildung 106).

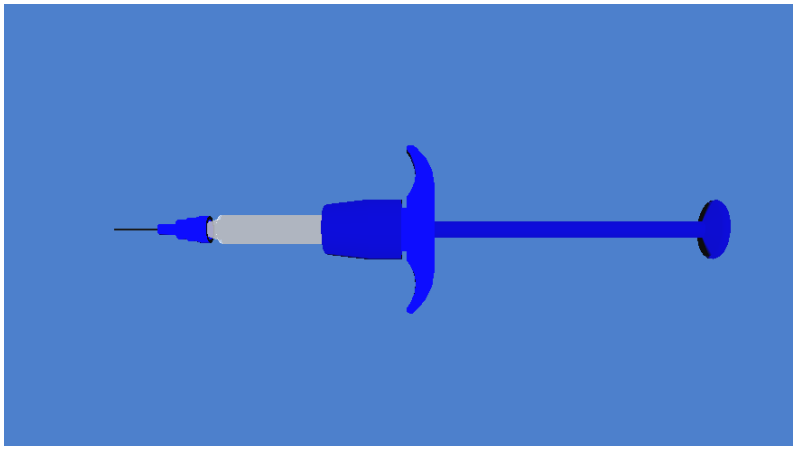


Abbildung 106 Werkzeug “Spritze”

Das Auf- oder Einbringen von Stoffen auf Oberflächen wird durch eine Metapher „Spraydose“ realisiert (siehe Abbildung 107.

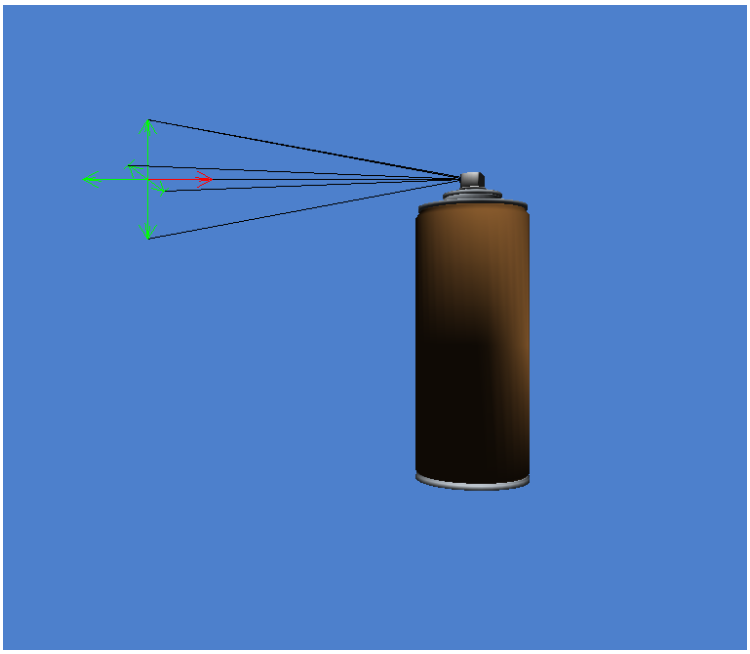


Abbildung 107 Werkzeug “Spraydose”

Jede der Metaphern wird durch ein entsprechendes 3-D-Tool repräsentiert.

Die bereitgestellten Funktionalitäten teilen sich entsprechend in drei Gruppen:

- Löschen:
    - Picking aktivieren/deaktivieren

  - Löschen der ausgewählten Objekte

- allgemeine Operationen zum Einfügen:
  - Tool in Sicht zentrieren: Diese Option zentriert das aktive Tool in der aktuellen Sicht des Benutzers in einer voreingestellten Entfernung.
  - Maussteuerung für das Tool aktivieren/deaktivieren: Wird die Anwendung mit einem 2-D-Eingabegerät, etwa einer Maus, bedient, kann mit dieser Funktion zwischen Navigationsmodus und Toolsteuerung umgeschaltet werden.
  - Rotation des Tools: Die Position des Werkzeugs wird fixiert, die Richtung dagegen bleibt frei.
- toolspezifische Operationen
  - Auswahl des Werkzeugs *Injektionsspritze*
  - Auswahl des Werkzeugs *Spraydose*
  - Einfügen: Ist die *Injektionsspritze* ausgewählt, wird innerhalb eines voreingestellten Volumens vor der Spitze der Injektionsspritze der ausgewählte Stoff eingefügt. Die jeweilige Menge ist einstellbar.
  - Einfügen: Ist die *Spraydose* ausgewählt, so wird in Richtung des Werkzeugs, der gewählte Stoff in gewählter Menge auf eine Oberfläche eingebracht.
  - Steuerung der Sprayachsen: Die Projektionsfläche des *Spraydosen*werkzeugs ist in der Standardeinstellung kreisförmig. Über die Auswahl der Achsen der Projektionsfläche lässt sich diese Fläche konfigurieren.

Die Abbildung 108 und Abbildung 109 verdeutlichen die Funktionen.

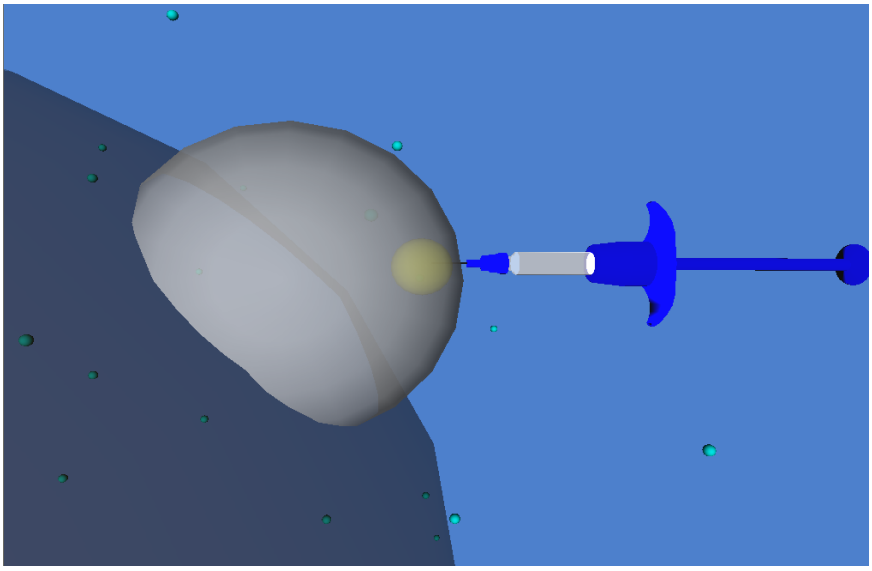


Abbildung 108 “Injektion”

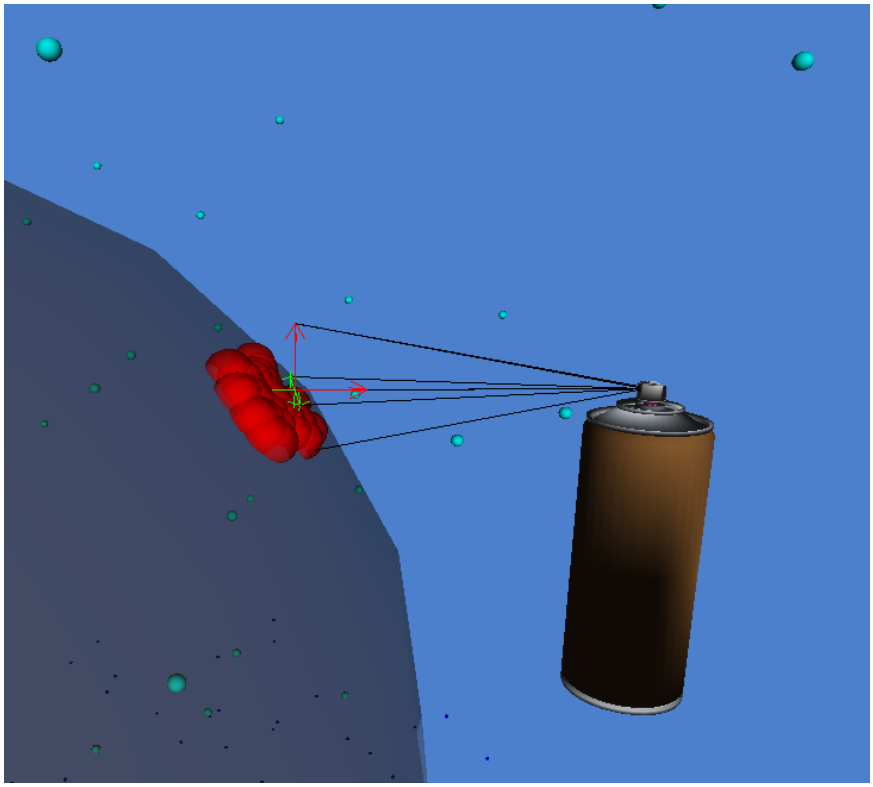


Abbildung 109 “Sprühen”

5.2.3.2.2 Raster

Der zweite wichtige Aspekt zur Vorbereitung eines Simulationsexperiments besteht nach der Philosophie dieser Arbeit in der Konfiguration des Reaktionsraums (der die gesamte Zelle oder Teile der Zelle enthält) für die Simulation. Für die Aufteilung des Raums gibt es verschiedene Strategien, als Beispiel wurde eine reguläre Gitterstruktur als Grundstruktur gewählt. Mit Hilfe eines Octree-Verfahrens [69] lässt sich die Rasterstruktur dynamisch adaptieren. Als Ordnungskriterium dient hier die Anzahl von Elementen pro Rastervolumen.

Die Operationen *Autogrid* und *Activate/Deactivate Grid* steuern diese Funktionen. *Autogrid* bewirkt eine Neuberechung des kubischen Einteilungsgitters anhand einer vordefinierten Anzahl von Elementen, die in einem Würfel enthalten sein dürfen.

Mit der Operation *Activate/Deactivate Grid* wird das Raster dargestellt oder ausgeblendet. Abbildung 110 zeigt das Raster.

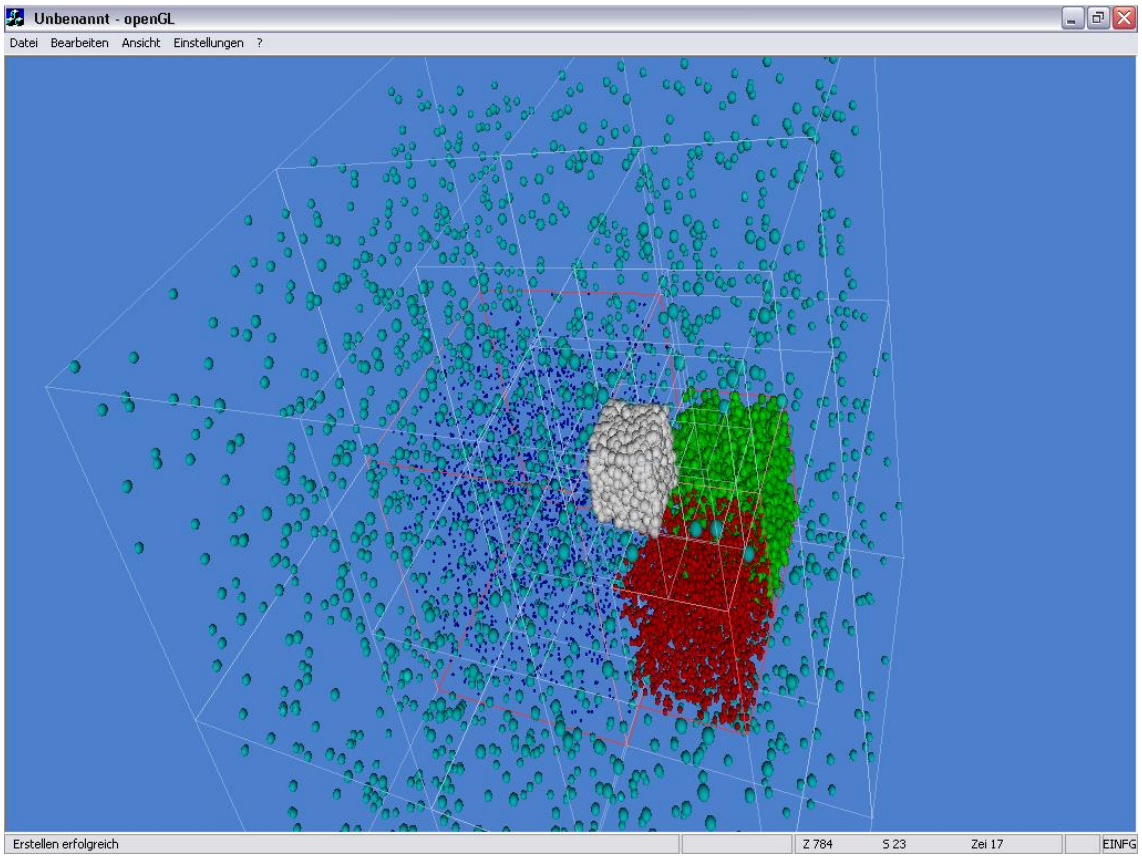


Abbildung 110 3-D-Grid

Die Anwendung wurde, wie auch der Zelleditor, in C++ implementiert. Dabei fand auch hier die OpenSceneGraph-API als 3-D-Bibliothek Anwendung. In einer erweiterten Fassung wird die Gitterstruktur auf nicht-reguläre Strukturen erweitert.

5.2.3.3 Gekoppelte Simulatoren: Diffusions-/ Reaktionssystem

Die vorgestellten Werkzeuge wurden genutzt, um beispielhaft ein System durch Agenten gekoppelter Simulatoren aufzubauen. Die hier verwendeten Simulatoren sind Implementierungen des von Gibson vorgestellten stochastischen *Next-Reaction*-Algorithmus [145] für Reaktionsnetzwerke. Gibson macht die Validität seines Algorithmus von einigen Randbedingungen abhängig, unter anderem von einer homogenen Durchmischung des Reaktionsvolumens. Er gibt also keine Hinweise zur Modellierung von Diffusion. Der Gibson-Algorithmus wurde nun um eine Diffusionskomponente erweitert. Der Grundgedanke ist hier, ein System gekoppelter Gibson-Simulatoren zu schaffen, deren Schnittstelle der Austausch von Stoffen durch Diffusion über die Grenzen zwischen zwei Reaktionsvolumen ist (siehe Abbildung 111).

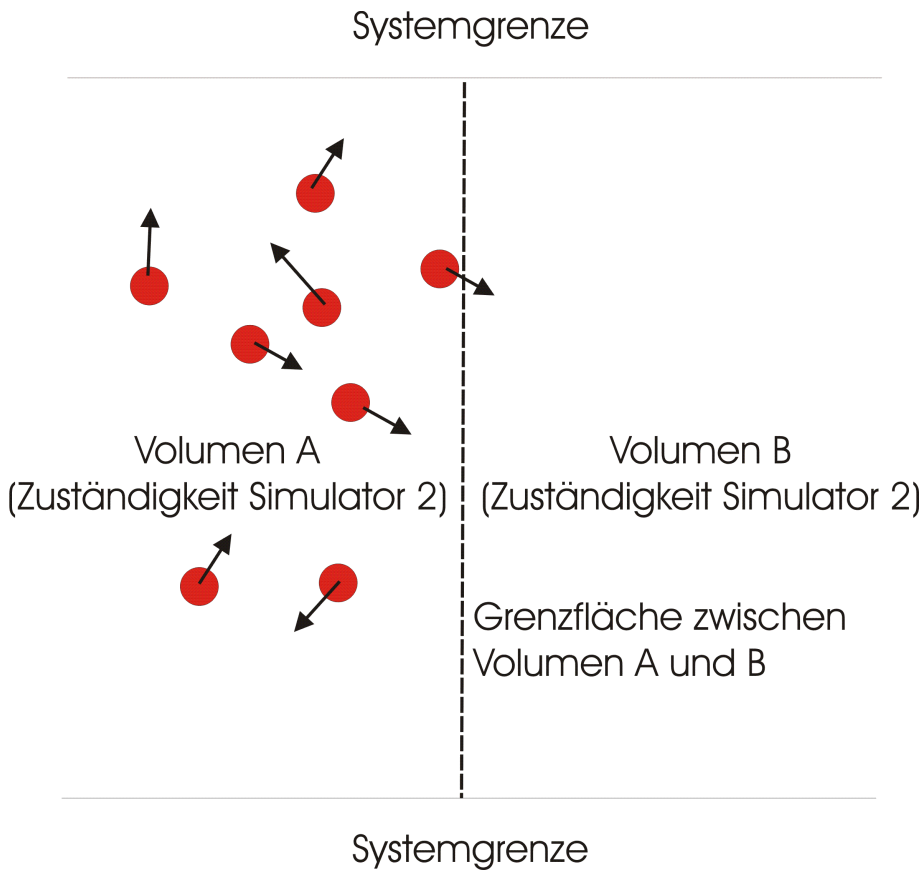


Abbildung 111 Grenze zwischen Simulationsvolumina

Der Austausch wird dabei über das oben beschriebene Protokoll durchgeführt. Es ist zu beachten, dass der Stoffaustausch von der Art des Simulators entkoppelt und der Simulator somit austauschbar ist.

Die hier vorgeschlagene Konfiguration geht von einem beliebigen, aber festen Zeitinkrement Δt aus; die weitere Annahme geht davon aus, dass Diffusion nur am Ende eines Zeitintervalls stattfindet. Diese Einschränkung ist nicht durch das vorgestellte Konzept vorgegeben, sondern vereinfacht die Parallelisierung der Simulation.

Der Gibson-Algorithmus soll hier nur in seiner Grundidee erläutert werden. Gegeben sind jeweils eine Anzahl n von Reaktionen und eine Anzahl von Molekülen, die entweder als Edukt oder als Produkt an den gegebenen Reaktionen teilnehmen. Die Grundannahme sowohl des Gillespie- als auch des Gibson-Algorithmus ist die Erkenntnis, dass sich die Wahrscheinlichkeit, welche aller möglichen Reaktion als nächste abläuft und wann diese Reaktion ablaufen wird, eindeutig berechnen lässt.

Mit Hilfe dieser berechenbaren Wahrscheinlichkeitsverteilung werden Zufallszahlen erzeugt, die den nächsten Reaktionszeitpunkt repräsentieren. Gibson verbessert die Methode, was die Anzahl der benötigten Zufallszahlen angeht, und kann so das Laufzeitverhalten des gesamten Verfahrens verbessern.

Der generelle Ablauf beider Verfahren ist etwa:

1. Berechnung der nächsten Reaktion
2. „Durchführen“ der Reaktion, also Verminderung der Edukte und Vermehrung der Produkte der Reaktion (Eine Reaktion der Form $A+B \Rightarrow C$ bedeutet, dass von der Anzahl der Moleküle vom Typ *A* und Typ *B* jeweils eins abgezogen wird, die Menge der Moleküle vom Typ *C* wird um eins erhöht.)
3. Neuberechnung der nächsten Reaktion unter Beachtung der Veränderungen, die sich durch die ausgeführte Reaktion ergeben haben
4. „Durchführen“...

etc.

Die bekannten Verfahren werden um Diffusionen erweitert. Hierbei wird der folgende Zusammenhang für die Anzahl *z* der Moleküle vom Typ *X*, die von Reaktionsvolumen *a* in das Reaktionsvolumen *b* diffundieren, angenommen:

$$z = \frac{D_X}{A_{a,b}} \#_{a,b} X$$

D_X ist der Diffusionskoeffizient der Moleküle vom Typ *X*.

$A_{a,b}$ ist der Flächeninhalt der Grenzfläche zwischen den Reaktionsvolumen *a* und *b*.

$\#_{a,b} X$ beschreibt die Anzahl der Moleküle vom Typ *X*, die theoretisch von *a* nach *b* diffundieren können. Dieser Wert berechnet sich aus der Gesamtanzahl *#X* der Moleküle vom Typ *X* im Ausgangsvolumen (*a*) wie folgt:

$$\#_{a,b} X = \min(\frac{mwl_X}{dist},1) \cdot \# X$$

mwl_X ist die mittlere Weglänge, die ein Partikel vom Typ *X* in einer Zeiteinheit (Δt) zurücklegt, sie berechnet sich als:

$$mwl_X = \sqrt{2 \cdot 3 \cdot D_X \cdot \Delta t}$$

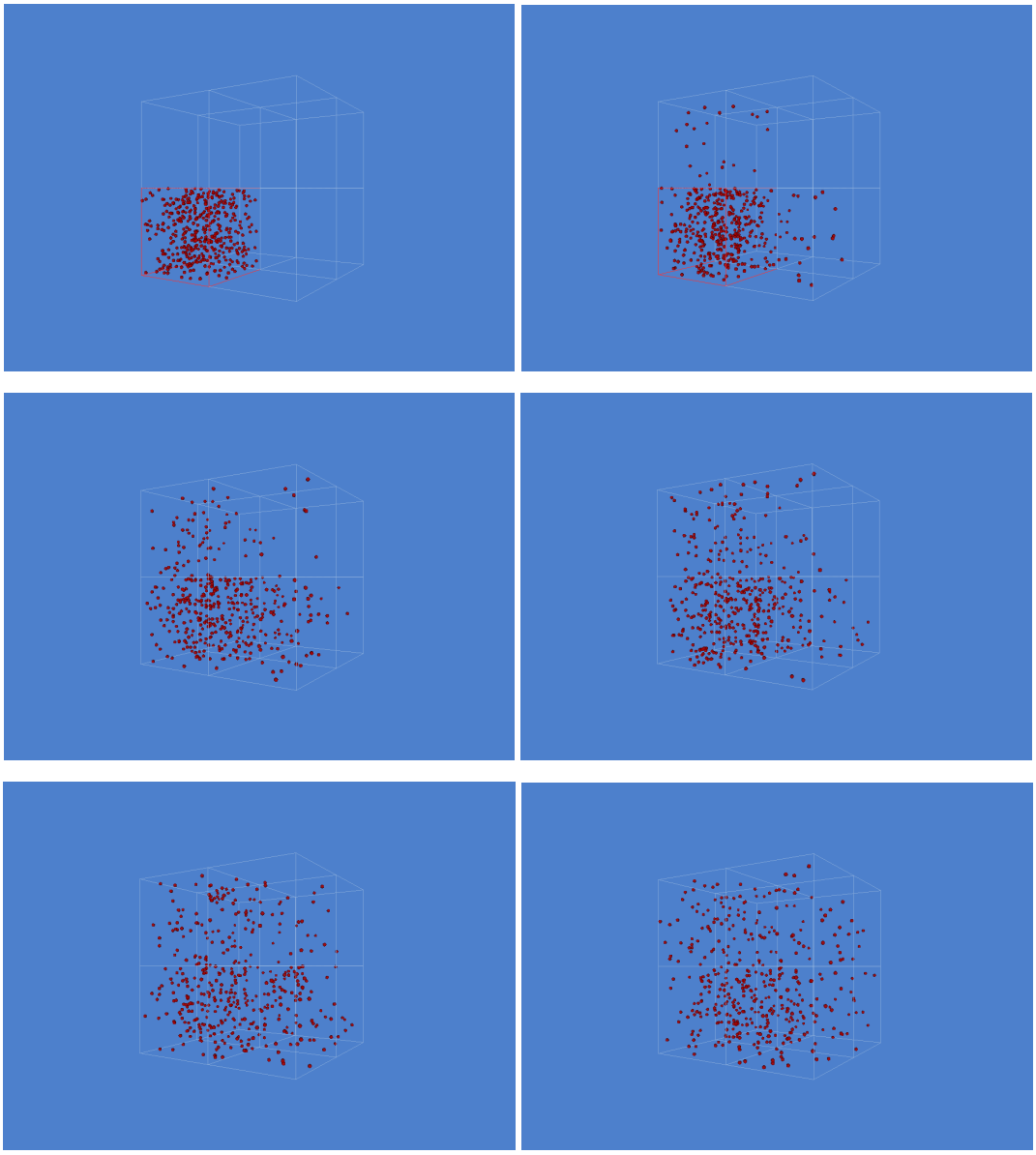
(Der Faktor 3 benennt die Dimensionalität des betrachteten Raums, hier *3*.)

dist ist der Abstand zwischen dem Schwerpunkt des Ausgangsvolumens und dem Mittelpunkt der Grenzfläche zwischen *a* und *b*.

Bisherige Veröffentlichungen zum Thema Diffusions-/ Reaktionssimulatoren (z. B. [157]) fordern ein reguläres Gitter der Reaktionsvolumina, also würfelförmige Reaktionsvolumina gleicher Abmessung. Der in dieser Arbeit beschriebene Ansatz vermeidet diese rigiden Anforderungen. Als einzige Anforderung an ein Reaktionsvolumen bleibt dabei die Forderung einer konvexen Geometrie. Diese Forderung ergibt sich aus dem Zusammenhang, den Quotienten der mittlerenWeglänge und des Abstands zwischen Schwerpunkt und Mittelpunkt der Grenzfläche als Maß zur Berechnung der Anzahl der möglicherweise an der Diffusion teilnehmenden Moleküle zu verwenden.

Die Sequenz in Abbildung 112 zeigt die Diffusion eines Stoffes in einem regulären Gitter von Reaktionsvolumina. Hier und in den folgenden Beispielen wird eine Struktur mit acht Volumina benutzt. Aus dieser Sequenz wird nebenbei ein Nachteil der regulären Gitter

deutlich: Diffusion findet nur über die Grenzflächen statt, also in die senkrecht und waagrecht benachbarten Volumen, nicht jedoch über die Diagonale.



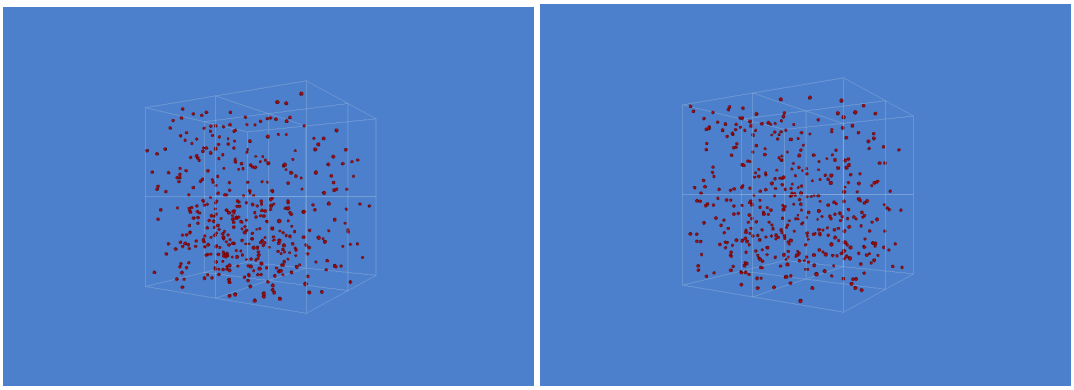
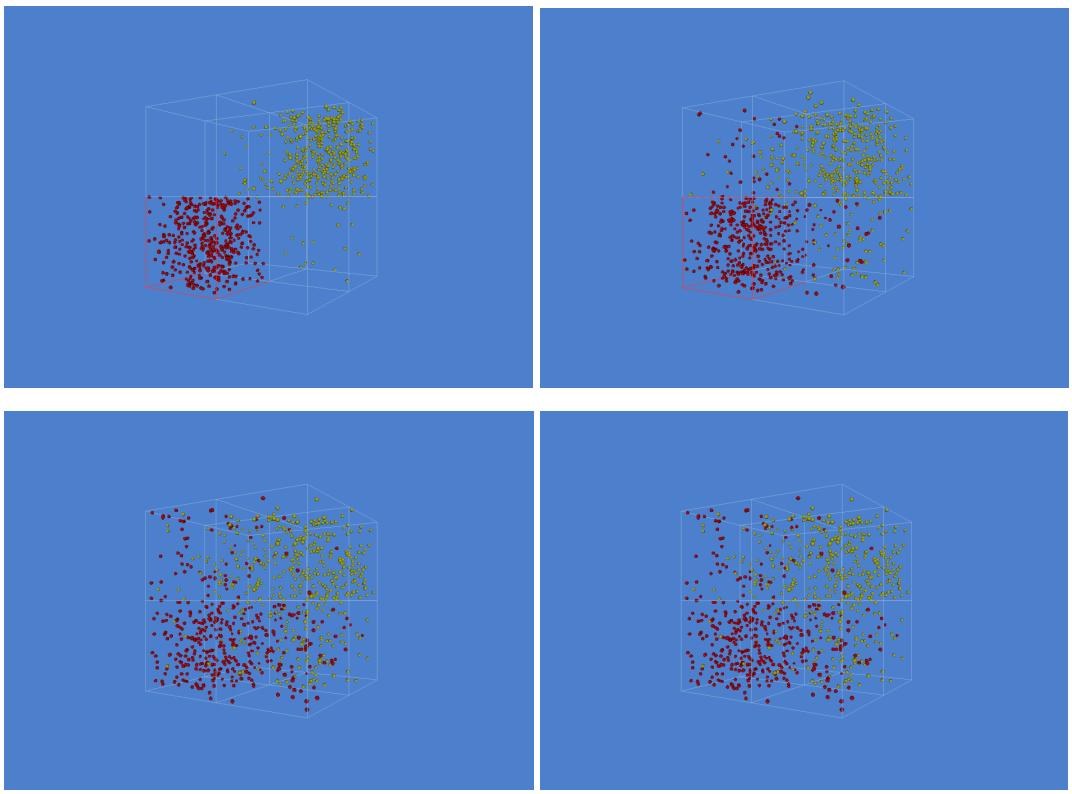


Abbildung 112 Diffusion eines Stoffes in 3D

In Abbildung 113 wird die Diffusion zweier Stoffe gezeigt, dargestellt durch rote (mittelgraue) und gelbe (hellgraue) Kugeln. Die beiden Species interagieren nicht miteinander, so dass es zu einer Durchmischung kommt.



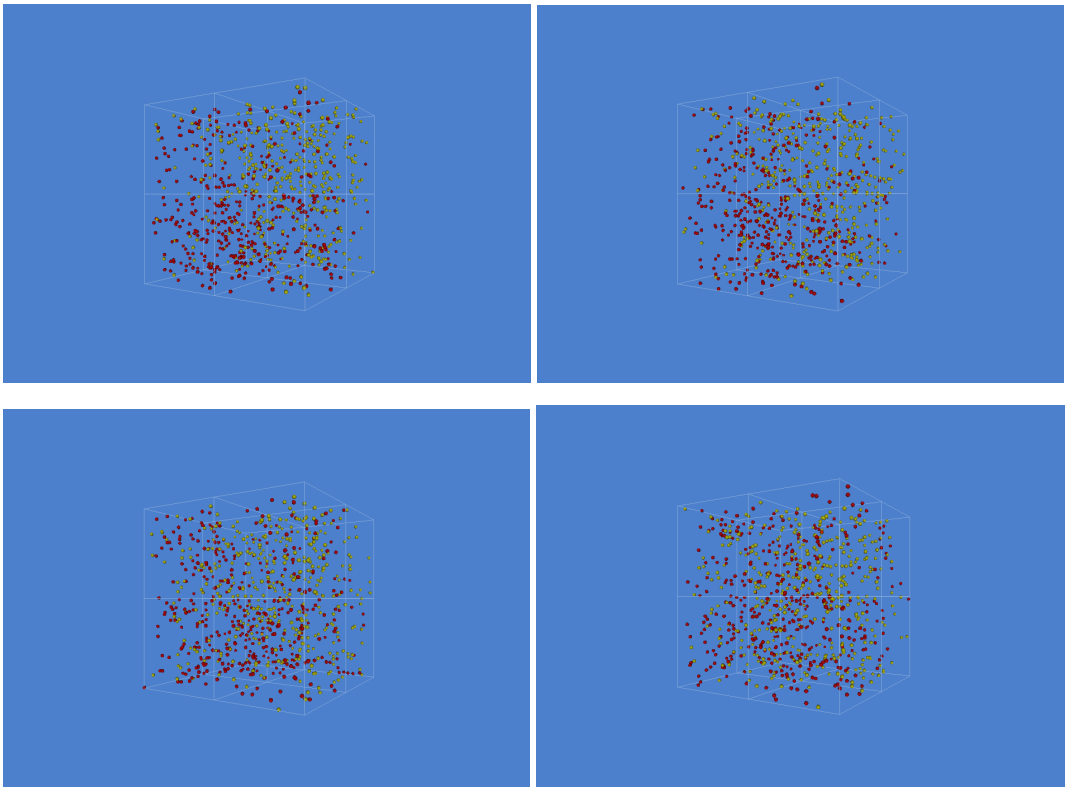
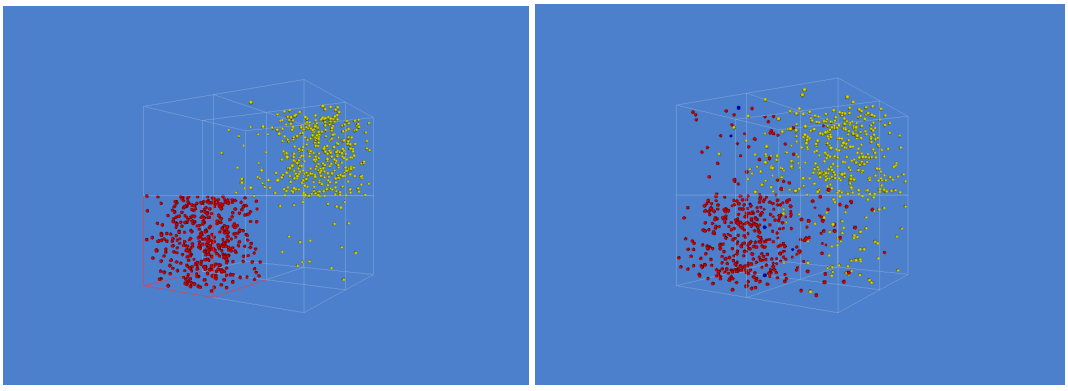


Abbildung 113 Diffusion zweier Stoffe, die nicht miteinander reagieren

Abbildung 114 schließlich zeigt ein einfaches Diffusions-/ Reaktionssystem. Wieder sind initial zwei Species vorhanden, wieder dargestellt durch rote (mittelgraue) und gelbe (hellgraue) Kugeln. Diesmal jedoch reagieren die beiden Stoffe miteinander in einer Reaktion, in der je ein Molekül jeder Sorte in ein Molekül einer neuen Species umgewandelt wird ($1A + 1B \Rightarrow 1C$). Die neue Species wird durch blaue (dunkelgraue) Kugeln visualisiert. Es ist deutlich zu sehen, wie sich die beiden Ausgangsspecies zunächst vermischen. Sofort setzt dann die Reaktion ein, bei der sich die Gesamtanzahl der Moleküle reduziert. Die Reaktionsgeschwindigkeit verringert sich mit der Abnahme der Menge der Eduktmoleküle.



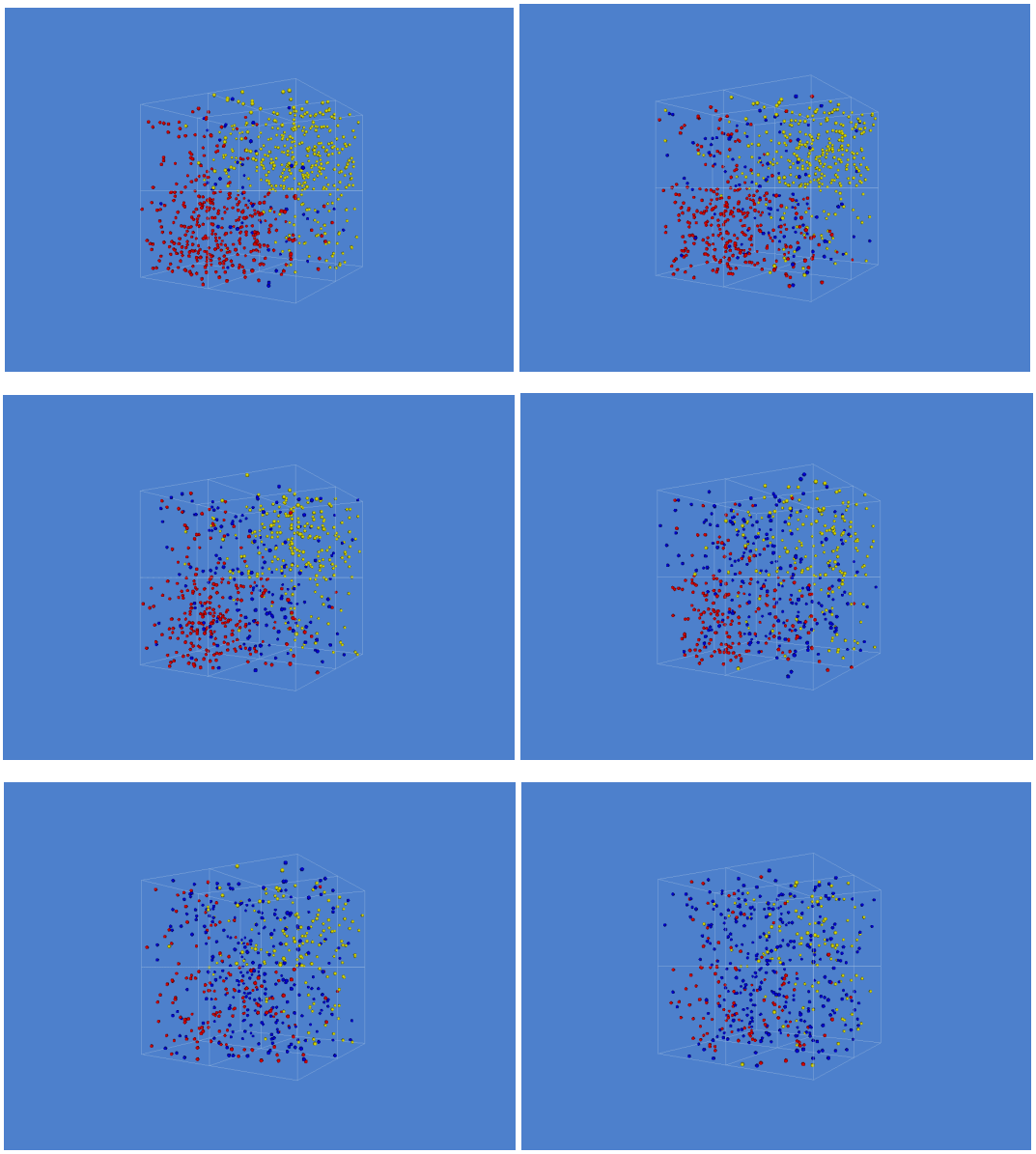


Abbildung 114 3-D-Reaktions-/ Diffusionssystem

Bei der Kopplung der Simulatoren wurde keine Annahme über die Art der verwendeten Simulatoren gemacht, mit einer Ausnahme: Dem von einem Simulator verwalteten Teilsystem müssen Moleküle entzogen werden (*diffuseOut*) und zugefügt werden können (*diffuseIn*).

5.2.4 Zelldifferenzierung

Auf Grundlage der vorgestellten Konzepte wurde der Prototyp eines Visualisierungssystem für Zelldifferenzierungsvorgänge, genannt *D-VISION,* entwickelt.

5.2.4.1 Das Datenproblem

Obwohl das Feld der Zelldifferenzierung in der biologische Forschung eine wichtige Rolle spielt, die nicht zuletzt durch die Diskussion über die ethischen Probleme der Forschung an menschlichen embryonalen Stammzellen einer breiteren Öffentlichkeit bekannt wurde, existieren bis jetzt nur wenige Arbeiten in Hinsicht auf Simulation und simulationsgeeignete Modellbildung [168].

Da insbesondere keine Modellierungen oder Simulationen von Zelldifferenzierungsvorgängen im dreidimensionalen Raum existieren, die als Grundlage für das zu erstellende Visualisierungssystem dienen können, müssen im Vorfeld erst einmal die Daten, die zur Visualisierung dienen sollen, spezifiziert werden. Gleichzeitig wird dadurch auch eine Anforderung an die von einem Simulator zu liefernden Daten formuliert. Die Datenspezifikation wurde auf Grundlage der in dieser Arbeit entwickelten Datenbeschreibungssprache entwickelt und ist als Vorschlag für künftige Simulatoren zu verstehen.

Als Nächstes folgt eine Analyse, welche Daten aus Sicht der biologischen Abläufe notwendig sind, wobei das Hauptaugenmerk auf der Darstellung der Zellen und ihres Zustands liegt.

Von den biologischen Grundlagen zum Datenmodell

Die Frage, welche Daten nötig sind, um Zelldifferenzierungsvorgänge zu visualisieren, soll im Folgenden erarbeitet werden. Da Zelldifferenzierung von Zellen vollzogen wird, sind es die Zellen, die visualisiert werden müssen.

Repräsentation von Zellen

Für die Repräsentation der Zellen sei auf die Zelleditoranwendung aus dem Bereich der Systembiologieanwendungen verwiesen. Allerdings liegt in dem dort beschriebenen System das Augenmerk auf dem Inneren der Zelle. Typischerweise wird in Anwendungen der Zelldifferenzierung und der Zellverbände dagegen der Blick von außen wichtiger.

Das äußere Erscheinungsbild von Zellen kann sehr stark in Form und Größe variieren. Dennoch sind vor allem in der frühen embryonalen Entwicklung die Zellen annähernd kugelförmig. Entsprechend können Zellen der Einfachheit halber als Kugeln repräsentiert werden, für deren Darstellung nur ein Mittelpunkt (also die Position) und eine Größe (Radius oder Volumen) notwendig sind. Sofern ein komplexeres Äußeres für die Repräsentation der Zellen angenommen wird, müssen die Daten dafür vom Simulator geliefert werden, d. h., der Simulator muss abgesehen von der Lage und der Größe auch die äußere Form der Zelle berechnen. Für die Darstellung von Differenzierungsvorgängen, bei denen es vor allem auf die Unterscheidung der Zelltypen ankommt, scheint diese ungleich schwierigere Berechnung unnötig zu sein.

Deshalb soll hier die Darstellung der Zelle mit Hilfe einer Kugel genügen.

Zellinneres

Gerade die Modellierung des Zellinneren liegt in der Domäne der Systembiologie; allerdings wird bei der Modellierung von Zelldifferenzierung häufig davon ausgegangen, dass die Bestandteile in den Zellen unbedeutend genug sind, um eine hohe Abstraktion bei der Simulation gerechtfertigt erscheinen zu lassen.

Die Zellorganellen sind zwar in den einzelnen Vorgängen in der Zelle involviert. Allerdings sind die Vorgänge an sich und nicht der exakte Ort innerhalb der Zelle, an dem sie ablaufen, relevant.

Das Innere einer Zelle muss folglich in dieser Anwendung zur Zelldifferenzierung nicht dargestellt werden.

Zellvorgänge

Es gibt Vorgänge in Zellen, die auch mikroskopisch beobachtet werden können. Zu diesen von außen sichtbaren Vorgängen zählen das Zellwachstum, die Zellteilung, Zellbewegungen und der Zelltod. Auch die Änderung der äußeren Form der Zelle ist beobachtbar. In der Zelle finden Vorgänge statt, die natürlicherweise nicht beobachtet werden können, wie die Synthese von Proteinen, die Verdoppelung der DNS oder der Abbau von Nährstoffen.

Da das Visualisierungssystem die Zelldifferenzierung darstellen soll, ergibt es keinen Sinn, die inneren Vorgänge zu visualisieren, da sie nur indirekt zur Zelldifferenzierung beitragen. Zwar ist die Anwesenheit eines Genregulators in einer Zelle mitentscheidend für die Gen-Expression der Zelle und damit für ihren Zelltyp, aber eine Visualisierung der Herstellung des Regulators scheint nicht erforderlich zu sein.

Die äußeren Vorgänge hingegen beeinflussen das Aussehen des von den Zellen gebildeten Gewebes. Da die Zellen in ihrer räumlichen Anordnung dargestellt werden sollen, müssen diese Vorgänge berücksichtigt werden. Im Folgenden wird nun analysiert, ob die bisherige Repräsentation einer Zelle durch eine Kugel (Position und Größe) dafür ausreicht.

Beim Zellwachstum, kann beobachtet werden, wie eine Zelle an Umfang zunimmt und manchmal ihre äußere Form verändert. Im Inneren werden durch Stoffwechselprozesse neue Substanzen hergestellt, die in die Zelle eingebaut werden. Beispielsweise werden Lipide erzeugt, die die Membranen der Organellen in der Zelle und die Zellmembran erweitern. Organellen wie die Mitochondrien vermehren sich durch Teilung. Es werden viele verschiedene Proteine synthetisiert, von denen einige das Cytoskelett der Zelle erweitern, andere sich in die Membranen einlagern usw. Da die Darstellung der inneren Prozesse sowie ein komplexes Äußeres für Zellen oben schon ausgeschlossen wurden, bleibt nur die Zunahme des Umfanges der Zelle, die visualisiert werden muss. Dazu reicht die Repräsentation durch eine Kugel aus, denn eine größere Kugel entspricht einer größer gewordenen Zelle.

Die Zellteilung bzw. der Zellzyklus einer Zelle besteht aus vielen Vorgängen, die als Phasen verstanden werden können. Es scheint klar zu sein, dass ein Simulator einige dieser Vorgänge berücksichtigen muss, um Zellteilungen und vor allem die Lage der neu entstehenden Zellen zu berechnen. Weiterhin müssen auch benachbarte Zellen und der Platz, der für eine Zellteilung vorhanden ist, berücksichtigt werden. Soll der Vorgang der Zellteilung visualisiert werden, so sind einige Daten notwendig. Da das Zellinnere nicht dargestellt werden soll, ist von der M-Phase nur die Cytokinese für die Visualisierung von Relevanz [168]. Die Zellfurchung (Einschnürung der Membran) und die Entstehung der beiden neuen Zellen respektive Kugeln müssen visualisiert werden. Dieser Aufwand scheint aber für ein Visualisierungssystem, das als Prämisse hat, die Zellzustände darzustellen, unnötig zu sein. Es reicht, die Zelle, die sich teilt, so lange als eine Kugel darzustellen, bis die Teilung vollzogen ist, und dann die beiden entstandenen Zellen durch zwei Kugeln an entsprechenden Positionen zu visualisieren.

Ein weiterer interessanter Aspekt der Zellteilung ist die Betrachtung der Verwandtschaftsverhältnisse von Zellen. Aus einer Zelle gehen bei der Zellteilung zwei Tochterzellen hervor. Dieses exponentielle Wachstum (ohne Beachtung des Zelltods) lässt sich als binärer Baum verstehen, in dem die Abstammung einer Zelle, die so genannte Zelllinie, zurückverfolgt werden kann.

Die Zellbewegung kann einfach als Änderung der Position der Kugel, die die Zelle repräsentiert, dargestellt werden.

Auch der Zelltod lässt sich mit der bisherigen Repräsentation einfach realisieren. Eine Zelle, die stirbt, löst sich auf, d. h., die Zellmembran bricht auf, und ihr Inneres ergießt sich in den extrazellulären Raum. Da das Zellinnere vom Visualisierungssystem nicht dargestellt werden soll, kann eine sterbende Zelle einfach durch Wegnahme der sie repräsentierenden Kugel erfolgen.

Eine Darstellung der Zelle als Kugel reicht aus, um auch äußere Zellvorgänge darzustellen. Die Information, in welche zwei Zellen sich eine Zelle teilt, ist für die Visualisierung ein interessanter Zusatz.

Zelltyp und Zellzustand

Der Zelltyp ist eine Benennung von bestimmten Zellen, die sich durch ihr Äußeres oder durch ihre Funktionsweise von anderen Zellen unterscheiden. Begründet ist diese Unterscheidung in der verschiedenen Gen-Expression der Zellen. Da alle Vorgänge in Zellen durch bestimmte Substanzen ausgelöst oder mit bestimmten Substanzen durchgeführt werden, definiert die Summe aller in der Zelle vorhandenen Substanzen den Zustand einer Zelle. Diese Substanzen werden entweder in der Zelle erzeugt oder von außen aus dem extrazellulären Raum über Transportmechanismen der Membran in die Zelle aufgenommen. Außerdem können Zellen sowohl über die Membran als auch mit Hilfe des Golgi-Apparates Substanzen aus dem Zellinneren in den extrazellulären Raum ausschütten.

Gene kodieren den Bauplan für Proteine. Ist ein Gen aktiv, so wird das entsprechende Protein in der Zelle synthetisiert. Proteine können Bausteine für die Zelle und ihre Bestandteile oder Katalysatoren für Reaktionen sein, die Substanzen in der Zelle umwandeln. Weiterhin können Proteine als Regulatoren fungieren, die bestimmte Vorgänge aktivieren, deaktivieren, fördern oder hemmen. Entsprechend sind Gene für die Funktionalität einer Zelle mitverantwortlich.

Insgesamt scheint der Zustand einer Zelle von den in ihr vorhandenen Substanzen, den sie umgebenden Substanzen und ihren aktiven bzw. inaktiven Genen abzuhängen.

Ein einfaches Visualisierungssystem könnte den von einem Zelldifferenzierungssimulator berechneten Zelltyp einer Zelle visualisieren, wobei alle möglichen Zelltypen eine wohl definierte Menge bilden. Eine solche Darstellung ist für eine Analyse der Zelldifferenzierung ungeeignet, da die dem Simulator zugrunde liegenden Entscheidungen über die Klassifikation der Zellen in Zelltypen nicht sichtbar gemacht werden können. Eine reine Auswertung des Zelltyps durch ein solches Visualisierungssystem würde dem Aufwand beim Modellieren und Simulieren nicht gerecht werden. Es scheint vielmehr sinnvoll, den Zellzustand zu visualisieren, d. h., abgesehen von der Zelle selbst soll auch der Zustand der Zelle visualisiert werden. Dazu müssen dem Visualisierungssystem folgende Daten vorliegen:

- die in einer Zelle aktiven Gene
- die Substanzen, die sich in einer Zelle befinden, sowie deren Menge (Konzentration)
- die Substanzen, die sich auf der Außenseite der Zelle befinden, sowie deren Menge
- der vom Simulator berechnete Zelltyp.

Als Substanzen werden hier nicht nur Proteine, sondern auch RNA, Co-Enzyme, Glucose oder alle anderen in Zellen vorkommende Stoffe bezeichnet.

Differenzierung

Es sollen Zelldifferenzierungsvorgänge visualisiert werden. Der eigentliche Vorgang der Zelldifferenzierung ist eine Zustandsänderung der Zelle. Die Funktionsweise der Zelle ändert sich dadurch, dass z. B. bisher inaktive Gene aktiv werden. Die Zelle wird Proteine herstellen,

die bisher nicht in ihr vorhanden waren. Diese Proteine können wiederum bisher nicht mögliche Reaktionen möglich machen oder sich an Stellen anlagern, an die bisher kein Protein gepasst hat. Hier spielt vor allem der zeitliche Ablauf eine Rolle. Die Differenzierung einer Zelle darzustellen bedeutet, ihren sich ändernden Zustand über eine Zeit hinweg aufzuzeigen. Ein Simulator berechnet die Zustände einer Zelle über die Zeit. Die Zeit als Komponente ist dabei wesentlich.

Die Daten der Zelle, d. h. ihre Position, ihre Größe und alle Daten, die ihren Zustand definieren, müssen sich auf einen konkreten Zeitpunkt beziehen. Differenzierung wird durch die fortlaufende Betrachtung über einen Zeitraum sichtbar.

Datenmodell

Diese Überlegungen führen zu dem in Abbildung 115 gezeigten Datenmodell.

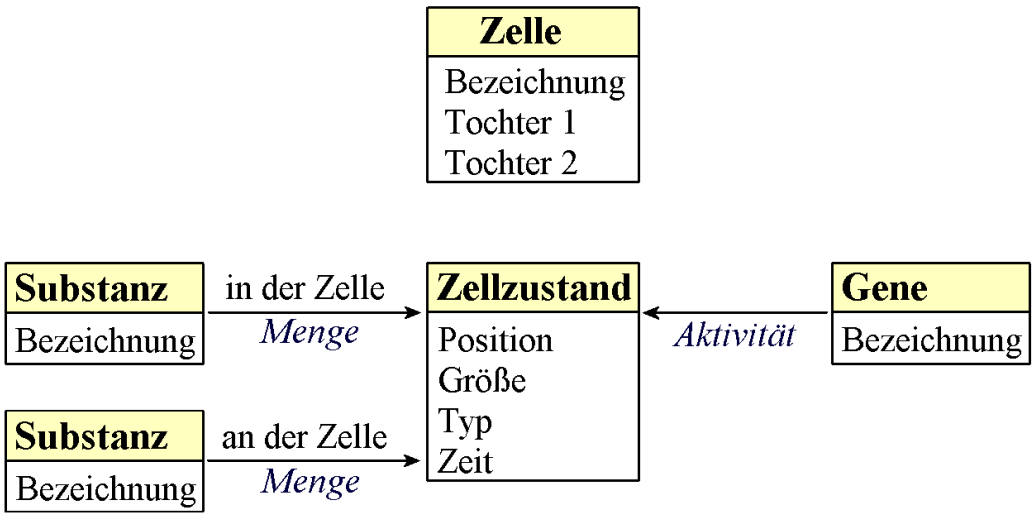


Abbildung 115 Konzeptionelles Datenmodell

Die Entität *Zelle* enthält als Attribut eine eindeutige Bezeichnung einer Zelle. Weiterhin enthält jede Zelle eine Referenz auf die beiden Tochterzellen. Aus diesen Informationen kann eine Baumstruktur der Verwandtschaften von Zellen extrahiert werden.

Die Entität *Zellzustand* stellt den Zustand einer Zelle zu einem bestimmten Zeitpunkt dar. Die Attribute sind: Zellposition, Zellgröße, Zelltyp sowie eine Zeitangabe. Weiterhin gehören zum Zustand einer Zelle die in ihr enthaltenen sowie die an ihrer Oberfläche vorhandenen Substanzen und die aktiven oder inaktiven Gene der Zelle. Diese Informationen werden über Verknüpfungen mit den Entitäten *Gen* und *Substanz* bereitgestellt. Die Entität *Gen* und die Entität *Substanz* enthalten jeweils als Attribut nur eine Bezeichnung. Durch die Verknüpfungen werden die Informationen des Zellzustandes, welche Gene aktiv oder inaktiv und welche Substanzen in welchen Mengen in oder an der Zelle vorkommen, abgebildet. Jeder Datensatz der Entität *Zellzustand* gehört zu einem konkreten Zeitpunkt.

Anforderungen an einen Simulator

Die im vorhergehenden Abschnitt definierten Daten, die für das Visualisieren von Zelldifferenzierungen verwendet werden sollen, müssen von einem Simulator geliefert werden.

Die hauptsächlich zu visualisierenden Daten sind die Zellen, die durch ihre Position und ihre Größe repräsentiert werden. Ein Simulator muss entsprechend in erster Linie die Lage und Größe von Zellen berechnen. Weiterhin sollen die Zelltypen bzw. die Zellzustände visualisiert werden. Entsprechend muss auch dies von einem Simulator geliefert werden. Um die vom Visualisierungssystem benötigten Daten berechnen zu können, muss ein Simulator Modelle verwenden, die eine entsprechende Berechnung ermöglichen. Ein Modell soll die biologische Wirklichkeit nachahmen.

Für *D-VISION* müssen Modelle die biologischen Vorgänge der Zelldifferenzierung auf einen dreidimensionalen Raum abbilden. Dieser Raum enthält neben den Zellen das extrazelluläre Medium. Die Zellen sind in sich geschlossene lokale Systeme, die untereinander und mit dem sie umgebenden extrazellulären Raum interagieren. Jede Zelle nimmt einen eigenen, sich nicht mit anderen Zellen überschneidenden Raum ein.

Zellen sind keine statischen Objekte, es sind aktive Einheiten, die sich mechanisch und biochemisch verändern. Um eine Zelle zu modellieren, müssen ihre Aktivitäten in das Modell einbezogen werden.

Zu den mechanischen Aktivitäten gehören Wachstum, Bewegung und Zellteilung. Neben diesen durch die Zelle selbst verursachten Veränderungen wirken auch von außen physikalische Kräfte auf sie ein, bedingt durch den Druck, der durch die veränderlichen Platzbedürfnisse benachbarter Zellen entsteht, sowie durch Zelladhäsion.

Die Vorgänge innerhalb der Zelle wurden bereits im Kapitel zur Systembiologie beschrieben. Interaktion zwischen Zellen findet durch Übergabe von Stoffen statt. Zellen können Substanzen direkt mit Nachbarzellen oder mit dem sie umgebenden Medium austauschen.

Neben den kontinuierlich in Zellen ablaufenden Prozessen müssen auch Ereignisse modelliert werden. Beispiele für Ereignisse sind die Aktivierung oder Deaktivierung eines Gens, die Zellteilung oder der Zelltod. Sie werden durch spezifische Auslöser hervorgerufen.

All diese unterschiedlichen Aspekte müssen durch Regeln in einem Gesamtmodell verknüpft werden, da die Mechanik und die Biochemie der Zellen miteinander in Wechselwirkung stehen.

Ein solches Modell stellt die Basis für eine Simulation von multizellulärer Entwicklung dar.

Das zeitliche Attribut der Daten, die *D-VISION* benötigt, ist diskret, d. h., die Position, die Größe und der Zustand der Zellen müssen sich immer auf einen konkreten Zeitpunkt beziehen. *D-VISION* generiert ein Standbild, eine Momentaufnahme vom simulierten Modell. Jede darzustellende Zelle muss dem Visualisierungssystem in einem Datensatz mit Position, Größe und Zustand vorliegen.

Die Arbeitsweise des Simulators ist davon unabhängig. Selbst wenn kontinuierliche Prozesse berechnet werden, werden nur die Daten von diskreten Zeitpunkten verwendet. Durch diese Vorgehensweise bleibt es dem Simulator freigestellt, mit Hilfe welcher methodischer Mittel er die Daten berechnet. Beispielsweise können alle Zellen als Gesamtheit oder jede Zelle einzeln nacheinander berechnet werden. Außerdem kann ein Simulator als verteiltes System konstruiert sein, in dem jede Zelle als Simulationseinheit behandelt wird.

Selbst Simulatoren, die nicht alle oben definierten Daten liefern, können mit *D-VISION* zusammenarbeiten. Die Minimalanforderung, die der Simulator erfüllen muss, damit noch ein brauchbares Bild erstellt werden kann, ist die Berechnung von Position und Größe der Zellen. Ohne diese Information können keine Zellen dargestellt werden.

Liegt dem Visualisierungssystem nur die Information zu Position und Größe der Zellen vor, wird es ein Abbild der Zellen ohne Unterscheidbarkeit der Zelltypen generieren. Jede weitere

Information, wie Zelltyp, aktive Gene, enthaltene oder umgebende Substanzen, ermöglicht eine Klassifikation der Zellen.

5.2.4.2 Datenfluss

Da das Visualisierungssystem unabhängig von Simulatoren einsetzbar sein soll, ist ein verteilter Aufbau des Gesamtsystems notwendig. Der gesamte Vorgang vom Erzeugen der Daten bis zum fertigen Bild wird von mehreren Programmen ausgeführt.

Von der Simulation zur Visualisierung

Ein Simulator benutzt Modelle, die ein System nachbilden, um Daten zu berechnen. Ein Visualisierungssystem stellt Daten optisch dar, um daraus Erkenntnisse zu gewinnen. Visualisierung wird eingesetzt, um die Daten eines Simulators zu analysieren. Man kann zwischen einer direkten Kopplung zwischen Simulator und Visualisierung und eigenständigen Systemen unterscheiden. Bei einer direkten Kopplung werden die Daten, die vom Simulator berechnet werden, direkt (in Echtzeit) dargestellt. Die Simulation und die Visualisierung sind eine Einheit: Wird die Simulation über Parameter beeinflusst, wirkt sich das auf die Visualisierung aus. Wird sie angehalten, so erhält man ein Standbild der Simulation. Sind allerdings Simulator und Visualisierungssystem unabhängig, so müssen die durch den Simulator berechneten Daten zwischengelagert werden. Zu einem anderen Zeitpunkt können sie dann vom Visualisierungssystem dargestellt werden. Allerdings ist dann keine Beeinflussung der Daten mehr möglich. Um andere Daten zu erhalten, ist ein erneuter, veränderter Simulationsdurchlauf erforderlich. Das Visualisierungssystem kann nur die Art und Weise der Darstellung (Mapping) verändern. Im ersten Fall wird eine Visualisierung für den Simulator auf Maß gefertigt. Im zweiten Fall ist es denkbar, ein vom Simulator unabhängiges Visualisierungssystem zu benutzen. Die Entscheidung, ob eine Visualisierung direkt oder indirekt gemacht wird, wird meist anhand der Dauer der Simulationsberechnung gefällt. Wenn ein Simulator sehr lange für seine Berechnungen braucht, so ist eine direkte Darstellung nicht sinnvoll. Es wird eine Zwischenspeicherung der Daten notwendig. Weiterhin ist es von Vorteil, dass auch ältere, gespeicherte Simulationsdurchläufe erneut visualisiert werden können. So wird auch ein Vergleich verschiedener Simulationen möglich.

Aufbau

Im vorliegenden Fall ist kein bestimmter Simulator definiert. Außerdem ist zu erwarten, dass ein Simulator, der die erforderlichen Daten liefert, aufgrund der Komplexität der Modelle zur Simulation von Zelldifferenzierungsvorgängen eine lange Berechnungszeit benötigt. Deshalb müssen die erzeugten Daten abgespeichert werden. Die Speicherung der Daten soll vor allem dem Visualisierungssystem dienen. Die Daten werden entsprechend in einer auf die Visualisierung zugeschnittenen, aufbereiteten Form gespeichert. Zwischen Visualisierung und Simulation vermittelt ein System, das für die Speicherung und Bereitstellung der Daten verantwortlich ist. Dieses im Folgenden *Aufbereiter* genannte System ist als System gekoppelter Agenten realisiert und verfügt über eine Datenbank, in der die aufbereiteten Daten der Simulationen zwischengespeichert werden. Der *Aufbereiter* stellt eine Schnittstelle zur Verfügung, mit der Daten in Form von XML-Dateien entgegengenommen werden. Soll vom *Aufbereiter* ein neuer Simulator unterstützt werden, der die Daten nicht über diese Schnittstelle liefert, so kann der *Aufbereiter* durch Programmierung dahingehend erweitert werden, dass es ihm möglich ist, die benötigten Daten aus den Resultaten des Simulators selbst zu extrahieren.

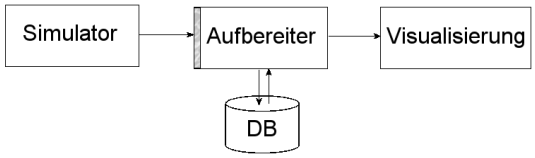


Abbildung 116 Systemaufbau D-Vision

Der Aufbau des Gesamtsystems ist in Abbildung 116 dargestellt. Neben den Komponenten *Simulator*, *Aufbereiter* und *Visualisierungssystem* ist auch der Datenfluss angegeben.

Der Datenfluss zwischen *Aufbereiter* und *Visualisierungssystem* wird durch Methodenaufrufe realisiert. Das *Visualisierungssystem* ruft dabei vom *Aufbereiter* zur Verfügung gestellte Methoden auf, die die gewünschten Daten zurückgeben.

Das so aufgebaute Gesamtsystem lässt verschiedene Architekturen für den Simulator zu. Er kann auch als verteiltes System oder als Agenten-System realisiert werden und somit die Vorteile von Grid-Computing nutzen. Einem auf mehreren Rechnern laufenden Simulator steht mehr Rechenkapazität zur Verfügung, und es können somit auch komplizierte Modelle berechnet werden. In so einem System besteht die Notwendigkeit einer Zentrale, die die Resultate der Simulation sammelt bzw. zur Verfügung stellt. Vorstellbar ist ein Computercluster, in dem jeder Computer einige wenige Zellen berechnet. Ein Zentralrechner verwaltet den Rechner-Pool, vergibt Ressourcen und koordiniert die Kommunikation zwischen den Einheiten.

Datenvisualisierung

Die Aufgabe des Visualisierungssystems ist es, die Daten, die von einem Simulator geliefert werden, zu visualisieren. Dazu ist es notwendig, die Daten auf optische Repräsentationen abzubilden.

Weiterhin soll ein Benutzer die dargestellten Daten nicht nur betrachten, sondern sie auch genauer untersuchen können. Eine Exploration der Daten muss vom Visualisierungssystem zugelassen werden.

Das Visualisierungssystem soll dem Benutzer Interaktionsmittel an die Hand geben, damit dieser in den Daten navigieren und sie analysieren kann.

5.2.4.3 Darstellung der Daten

Die Visualisierung soll ein dreidimensionales Bild der darzustellenden Zellen generieren, das die zu simulierenden Zellen räumlich nachbilden soll. Dazu wird ein virtueller dreidimensionaler Raum benutzt, in dem die virtuellen dreidimensionalen Objekte positioniert werden und der beim Rendering (Erzeugen des Bildes) auf den zweidimensionalen Bildschirm des Betrachters abgebildet wird. Hierzu wird eine virtuelle Kamera positioniert und das von ihr aufgefangene Bild erzeugt.

Zellen

Wie oben schon mehrfach erwähnt, liegt auf den Zellen das Hauptaugenmerk der Visualisierung. Die Daten, die eine Zelle repräsentieren, nämlich die Position und die Größe, müssen verwendet werden, um die Zellen darzustellen.

Geometrie

Es werden Kugeln zur Darstellung der Zellen verwendet; die Position der Zellen, die mit drei Koordinaten in den Daten angegeben sind, bestimmt den Mittelpunkt der Kugel.

Um verschiedene Simulatoren zu unterstützen, kann die Größe der Zelle, die in den Daten als ein Zahlenwert angegeben ist, als Radius oder als Volumen angegeben werden. Um die Größenverhältnisse der Zellen korrekt wiederzugeben, ist es notwendig zu wissen, ob es sich bei der Größe um den Radius oder um das Volumen einer Zelle handelt. Die Zahlenwerte können entsprechend umgerechnet werden.

Die Zellen werden als Kugel-Objekte im virtuellen dreidimensionalen Raum dargestellt, wobei sie nicht als massive Körper, sondern nur durch ihre Oberfläche, also als Hohlkörper, repräsentiert werden.

Ein Zellhaufen wird als Kugelhaufen dargestellt. Es obliegt dem Simulator, korrekte Werte für die Positionen und die Größe der Zellen zu liefern.

Da die Zellen je nach Simulator verschieden positioniert und skaliert sein können, ist es notwendig, vor der Darstellung der Zellen den insgesamt benötigten Raum zu bestimmen, um die Abbildungen in den vorhandenen virtuellen Raum einpassen zu können. Somit muss einem Simulator keine Vorgabe zu den Größenverhältnissen gemacht werden. Allerdings sollten die vom Simulator berechneten Größen und Positionen der Zellen zueinander passen.

Die Kugeln, die die Zellen repräsentieren, sollten sich möglichst nicht überschneiden, da sich Zellen in einem Organismus auch nicht überschneiden.

Die so entstehenden Bilder ähneln nur bedingt tatsächlichen Aufnahmen von Zellhaufen der frühen Embryogenese.

Die Zellen sehen tatsächlich eher wie sich gegenseitig deformierende Kugeln aus, z. B. wie mehrere Eidotter in einer runden Schüssel.

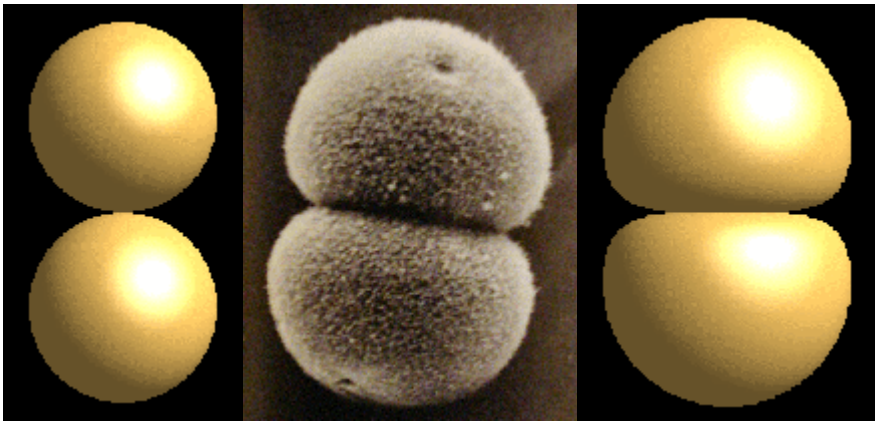


Abbildung 117 Zweizelliger Embryo mit virtueller Darstellung [170]

Es liegt nahe, das tatsächliche räumliche Aussehen der Zellen auch für die Visualisierung anzunähern, statt regelmäßige Kugeln zu verwenden. Diese Annäherung wird durch sich gegenseitig deformierende Kugeln erreicht.

Die Grundlage für eine Deformation zweier Kugeln ist, dass sie sich überschneiden. Eine einfache Realisierung dieser Deformation kann durch die Verschiebung der Punkte an der Oberfläche der Kugel im Überschneidungsbereich senkrecht zur Schnittebene erfolgen. Die so entstehenden Bilder entsprechen in ihrem Aussehen schon viel besser dem tatsächlichen Aussehen der Zellen (siehe Abbildung 117).

Mit der Darstellung der Zellen als deformierte Kugeln entstehen zwei Probleme:

Erstens müssen sich die Kugeln überschneiden, damit sie sich gegenseitig deformieren können. Wenn ein Simulator allerdings darauf achtet, dass sich die Kugeln nicht überschneiden, so wird er nur entsprechende Daten liefern. Eine Möglichkeit, trotzdem deformierte Kugeln darzustellen, entsteht, wenn die Größe aller Zellen mit einem benutzerdefinierten Faktor vergrößert wird, da sie sich dann überschneiden werden.

Das zweite Problem entsteht dadurch, dass sich die Größe, also das Volumen der Kugeln, bei der Deformation verkleinert. Um die Größe der Zellen korrekt wiederzugeben, muss diese Verkleinerung kompensiert werden. Es ist denkbar, dies z. B. durch eine Vergrößerung der Kugel vor der Deformation zu erreichen. Allerdings wird eine größere Kugel stärker deformiert und deformiert auch die angrenzenden Kugeln stärker. Ebenso wie mehrfache Deformationen einer Kugel führt dies dazu, dass die zur Kompensation benötigte Vergrößerung der Kugel nicht exakt berechnet werden kann. Durch mehrere Iterationen von Vergrößerung und anschließender Deformation kann eine Lösung für die Verkleinerung bei der Deformation gefunden werden. Voraussetzung dafür ist natürlich, dass die Daten des Simulators genug Platz für die Zellen lassen.

Mit der Darstellung der Zellen als deformierte Kugel stellt sich das Problem, dass die dargestellten Daten verändert werden könnten und damit die Expressivität der Visualisierung leidet. Allerdings ist der Nutzen durch eine Darstellung der Zellen als deformierte Kugel groß, da ein viel realistischeres Bild der Zellen erreicht werden kann.

Auch ein Simulator könnte mit dieser Form der Darstellung der Zellen rechnen und sich bei der Größe der Zellen auf das Volumen konzentrieren. Damit die Zellen nebeneinander genug Platz finden, muss der Simulator die Platzverhältnisse berücksichtigen.

Material

Da die Kugeln, die die Zellen repräsentieren, in einem virtuellen, dreidimensionalen Raum abgebildet werden sollen, stellt sich die Frage nach ihrer Oberflächengestaltung. Die Oberfläche der Kugeln soll matt und einfarbig sein. Durch die Beleuchtungsrechnung beim Rendering wird ein Eindruck von Räumlichkeit in der zweidimensionalen Abbildung durch Schattierungen und Lichtreflexionen erweckt. Texturen, Mehrfarbigkeit oder sonstige Oberflächenmuster sind nicht zweckmäßig, da sie eine differenzierte Zelloberfläche suggerieren könnten. Allerdings können solche Oberflächengestaltungen eine Identifikation der Zelle ermöglichen. Dies ist auch durch eine Einfärbung der Oberfläche mit einer anderen Farbe möglich. Da Farbe eine preattentive optische Eigenschaft ist, die eine sehr schnelle Identifikation erlaubt [169], soll diese einfache Variante der Markierung von bestimmten Zellen verwendet werden. Durch Einfärbung einiger Zellen können diese im Zellhaufen aufgespürt und identifiziert werden. Eine Markierung von bestimmten Zellen wird möglich, da sich die andersfarbigen Zellen vom Rest abheben.

Da ein Zellhaufen innen liegende Zellen enthält, die von äußeren verdeckt sein können, ist es sinnvoll, eine Möglichkeit bei der Darstellung von Zellen einzubauen, die einen Einblick ins

Innere gewährt. Dies kann durch eine transparente Gestaltung der Oberfläche erreicht werden. Durch einen transparenten Zellhaufen kann hindurchgeblickt werden, und es wird erkennbar, wie die Zellen im Inneren angeordnet sind.

Auch 2D

Eine andere Möglichkeit, den Blick ins Innere eines Zellhaufens freizugeben, ist es, einen Schnitt durch den Zellhaufen als 2-D-Bild darzustellen. Das so entstehende Bild ähnelt in seinem Aussehen dem Anblick von Zellen durch ein Mikroskop.

Die kugelförmigen Zellen werden in einem solchen Schnittbild als Kreise angezeigt. Eine durchschnittene deformierte Kugel wird in der zweidimensionalen Darstellung ein deformierter Kreis sein.

Diese Methode ermöglicht ein sehr viel exakteres Bild vom Inneren eines Zellhaufens, da die vor oder hinter der Schnittebene liegenden Zellen nicht angezeigt werden.

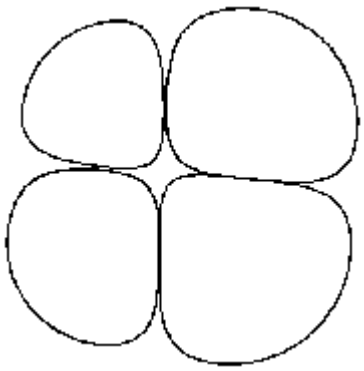


Abbildung 118 2-D-Schnitt

Das zweidimensionale Bild (Abbildung 118) ergibt sich durch einen Schnitt durch die dreidimensionale Darstellung. Die zur Darstellung des zweidimensionalen Bildes notwendigen Daten werden aus den dreidimensionalen Daten der Zellen und der Lage der Schnittebene berechnet.

Genauso wie in der dreidimensionalen Darstellung werden auch in der zweidimensionalen Darstellung die Zellen durch ihre Hüllen repräsentiert. Im 2-D-Bild entsprechen die Hüllen den geschlossenen Linien (Kreisen oder deformierten Kreisen). Auch im Schnittbild können Zellen über eine Einfärbung der Hülle markiert werden. Dies stellt ein Unterscheidungskriterium der Zellen dar. Darüber hinaus kann auch das Innere einer Zelle eingefärbt werden. Diese Einfärbung des Inneren der Zelle kann für die Visualisierung anderer Daten (siehe unten) verwendet werden.

Substanzen und Gene

Im Datenmodell sind Substanzen und Gene in den Zellen enthalten. Ihre Menge bzw. ihr Status gehört zu den Daten der Zelle. An der Oberfläche der Zellen befinden sich ebenfalls Substanzen, deren Menge ebenso zu den Daten von Zellen gehört. Sie repräsentieren die Konzentration einer Substanz im extrazellulären Raum, die in direktem Kontakt mit der Zelloberfläche stehen. Die Substanzen bzw. ihre Menge in oder an der Zelle sowie die Gene, die in der Zelle aktiv oder inaktiv sind, sind raumbezogene Daten, die direkt oder auch indirekt angezeigt werden können. Eine direkte visuelle Repräsentation all dieser Daten in der Abbildung der Zellen würde das Bild überladen. Die Daten werden stattdessen indirekt

dargestellt, indem nach der Selektion einer Zelle die zu dieser Zelle gehörenden Daten in einer separaten Anzeige aufgelistet werden. Dabei werden die Substanzen, die sich in dieser Zelle befinden, mit ihrer Bezeichnung und ihrer Menge in einer Liste aufgeführt. Ebenso wird eine Liste für die Substanzen und deren Konzentrationen, die sich an der Oberfläche der selektierten Zelle befinden, angegeben. In einer dritten Liste werden die Gene, die sich in der markierten Zelle befinden, sowie die Information, ob dieses Gen in der Zelle aktiv oder inaktiv ist, aufgeschlüsselt.

Abgesehen von den Listen, die den Zellinhalt einer Zelle anzeigen, sind weitere Listen für Substanzen und Gene sinnvoll: eine Liste, in der alle in der darzustellenden Simulation vorkommenden Substanzen mit ihren Bezeichnungen aufgezählt sind, und eine Liste, in der alle Gene, die von der Simulation berücksichtigt werden, enthalten sind. Diese beiden Listen ermöglichen einen Überblick über die Substanzen und Gene, die insgesamt als Zellbestandteil in Frage kommen.

Eine Substanz in der Zelle

Ein interessanter Aspekt der Substanzen ist ihre räumliche Verteilung, d. h. ihre Konzentration über die Zellen hinweg. In diesem Zusammenhang ist es sinnvoll, den Konzentrationsgradienten einer Substanz in der räumlichen Anordnung der Zellen sichtbar zu machen. Es wird dadurch möglich, einen Zusammenhang zwischen der Konzentration einer Substanz und dem Zustand einer Zelle zu erkennen. Hierzu wird zunächst eine Substanz ausgewählt, deren Konzentrationsgefälle angezeigt werden soll. In den Zellen wird nun die Konzentration dieser Substanz visuell dargestellt. Die Darstellung der Substanz in den Zellen soll nicht als Visualisierung des Inhaltes einer Zelle, sondern als Darstellung der Konzentration einer bestimmten Substanz über alle Zellen hinweg verstanden werden. Die Repräsentation der Konzentration einer Substanz wird über Objekte realisiert, die sich im Innern der Zellen befinden.

Eine in [171] vorgestellte Methode zur Darstellung von Volumendaten stand für die Idee der Darstellung des Konzentrationsgefälles einer Substanz mit Hilfe kleiner Objekte Pate. Die Daten werden bei [171] über regelmäßig im Raum platzierte kleine Objekte repräsentiert, wie z. B. Kugeln oder Würfel – Tiny Cubes (TC). Sie werden entsprechend dem Konzentrationswert an ihrer Position eingefärbt. Diese Darstellung bietet den Vorteil, dass durch die Zwischenräume der Objekte hindurch weiter hinten liegende Objekte betrachtet werden können (siehe Abbildung 119).

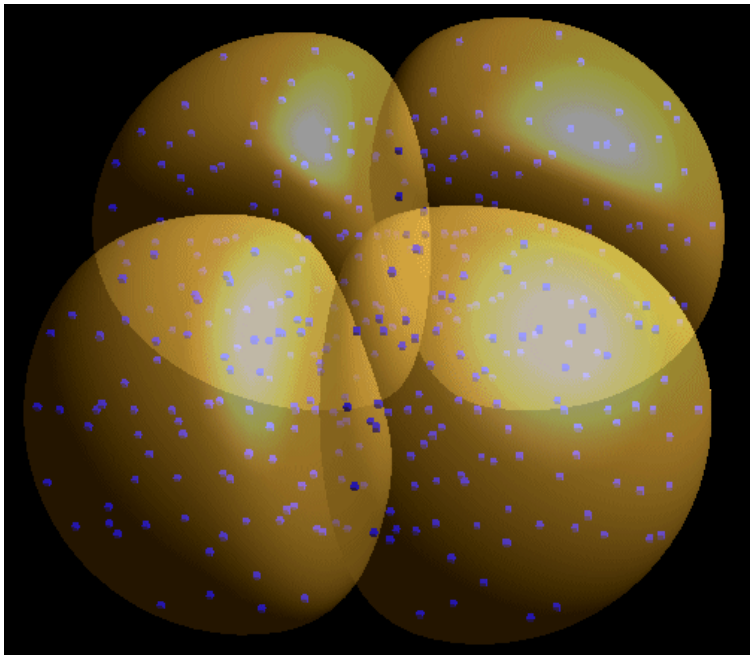


Abbildung 119 Zelldarstellung mit einem Inhaltsstoff

Anders als bei der in [171] beschriebenen Methode sollen die TC jedoch nicht regelmäßig platziert und nicht entsprechend dem darzustellenden Wert eingefärbt werden. Die TC sind gleichfarbig und sollen allein durch die Anzahl, mit der sie in einer Zelle präsent sind, die Konzentration einer Substanz in dieser Zelle darstellen. Der Vorteil der Methode, durch die Zwischenräume einen Einblick auf die Konzentration in allen Zellen zu erhalten, bleibt bestehen. Damit kann ein Eindruck von der Verteilung dieser Substanz im gesamten Raum gewonnen werden. An Orten, an denen sich TC häufen, ist die Konzentration größer als in Bereichen mit einer weniger dichten TC-Verteilung.

Da im Datenmodell die Menge einer Substanz durch einen Zahlenwert angegeben ist, kann dieser Wert je nach Simulator die Konzentration oder die Stoffmenge der Substanz in der Zelle sein. Es ist notwendig zu wissen, ob es sich um Konzentration oder um Stoffmenge handelt, denn für die Darstellung mit der oben beschriebenen Methode ist die Stoffmenge relevant. Eine Konzentration kann unter Berücksichtigung des Zellvolumens in Stoffmenge umgerechnet werden.

Es wird also die Stoffmenge einer Substanz in den Zellen dargestellt. Dazu werden entsprechend viele TC in einer Zelle platziert. Die Zellen müssen transparent sein, um die TC sehen zu können. Die Anzahl der TC, die eine bestimmte Stoffmenge einer Substanz in einer Zelle repräsentieren, wird durch eine Skalierung mit Hilfe der maximal vorkommenden Stoffkonzentration und der maximal darstellbaren TC in einer Zelle errechnet. Die Konzentration der Substanz ergibt sich optisch, denn viele TC auf kleinem Raum entsprechen einer hohen Konzentration.

Die Darstellung des Konzentrationsgefälles auf diese Art und Weise ist nicht als exakte Datenvisualisierung zu verstehen. Da nur eine begrenzte Anzahl an TC in einer Zelle sinnvoll ist, weil zu viele Objekte den Blick ins Innere des Zellhaufens und damit den Vorteil dieser Darstellung zunichte machen würden, wird die Konzentration stark gerundet wiedergegeben.

Kleine Unterschiede in der Konzentration sind unter Umständen nicht visuell erkennbar. Um dies zu verbessern, könnte die Skalierung der darzustellenden Stoffmengen vom Betrachter eingestellt werden. Allerdings kann es dann vorkommen, dass zur Darstellung der Stoffmenge in einer bestimmten Zelle nicht genügend TC in dieser Zelle Platz finden, also die Menge der Substanz in dieser Zelle nicht richtig wiedergegeben werden kann. Werden solche überladenen Zellen vom Betrachter in Kauf genommen, kann er durch Änderung der Skalierung auch noch kleine Unterschiede der Stoffmenge in anderen Zellen sichtbar machen.

Insgesamt ist die Darstellung des Konzentrationsgefälles einer Substanz als Methode gedacht, um beim Betrachter optisch einen Eindruck der Substanzverteilung hervorzurufen und damit Zusammenhänge erkennbar werden zu lassen. Zur genauen Datenexploration werden die Mengen der Substanzen, wie oben beschrieben, für jede Zelle numerisch in einer Liste wiedergegeben.

Sinnvoll kann es auch sein, das Konzentrationsgefälle von mehr als einer Substanz sichtbar zu machen. Dazu wird jede Substanz durch verschiedenfarbige TC repräsentiert. Um tatsächlich noch Zusammenhänge erkennen zu können, scheinen allerdings mehr als drei Substanzen nicht sinnvoll, da sonst die Zellen überladen sind und zusätzlich das Unterscheiden der Farben erschwert wird. Für die unterschiedlichen Substanzen sollten möglichst verschiedene Farben wie Rot, Blau und Grün verwendet werden, damit eine Unterscheidung der unter Umständen sehr kleinen TC möglich ist.

Wird mehr als ein Konzentrationsgefälle dargestellt, stehen in den Zellen auch weniger Positionen für die TC zur Verfügung. Dies hat eine stärkere Quantisierung der Werte zur Folge.

Eine Substanz außerhalb der Zelle

Genauso wie die Konzentration der in den Zellen enthaltenen Substanzen im Zusammenhang mit dem Zelltyp steht, können auch die an der Oberfläche der Zelle vorhandenen Substanzen den Zelltyp beeinflussen. Um auch diese Zusammenhänge optisch erkennbar werden zu lassen, ist es ebenso sinnvoll, den Konzentrationsgradienten der Substanzen im extrazellulären Raum sichtbar zu machen. Da der extrazelluläre Raum nicht direkt, sondern nur indirekt über die Oberfläche der Zellen in den Daten repräsentiert wird, muss die Konzentration einer Substanz im extrazellulären Raum an den Oberflächen der Zellen dargestellt werden.

Die Konzentration einer Substanz wird über das Einfärben der Zelloberfläche realisiert, da bei dem vorgeschlagenen Datenmodell für die gesamte Oberfläche einer Zelle dieselbe Konzentration einer Substanz angegeben ist. Die Farbe der Zelle ergibt sich über einen Farbgradienten, mit dem die Konzentration auf eine bestimmte Farbe einer Farbskala abgebildet werden kann. Bei verschiedenfarbigen Zellen liegen an den Oberflächen verschiedene Konzentrationen der Substanz vor, deren Konzentrationsgefälle angezeigt wird.

Damit auch die Einfärbungen der innen liegenden Zellen sichtbar werden können, müssen die Zellen transparent dargestellt werden.

Als Farbgradient können die im Allgemeinen als natürlich empfundene Farb-Temperatur-Skala oder die Magenta-Skala, die die menschliche Farbempfindlichkeit für Violetttönungen ausnutzt, dienen. Eine Abbildung der Skala ist notwendig, um eine optische Einordnung der dargestellten Konzentration vornehmen zu können.

Mit dieser Art der Darstellung kann nicht mehr als das Konzentrationsgefälle einer einzelnen Substanz im extrazellulären Raum dargestellt werden. Die Anzeige der Konzentrationen mehrerer Substanzen ist somit nicht möglich.

Durch die Art der Darstellung der Konzentrationen von Substanzen in einer Zelle und an deren Oberfläche ist es möglich, beides zu kombinieren. In *D-VISION* können also die

Konzentrationsgradienten von maximal drei Substanzen in den Zellen und einer Substanz im extrazellulären Raum dargestellt werden. Die Darstellung der Konzentrationsgefälle mehrerer Substanzen ermöglicht es, nicht nur Zusammenhänge zwischen Substanz und Zellzustand, sondern auch eventuelle Zusammenhänge zwischen den Substanzen sichtbar zu machen.

Gene

Neben den Substanzen liefern die Daten vom Simulator auch Informationen über die in den Zellen enthaltenen Gene und deren Aktivität. Neben der oben beschriebenen Möglichkeit, sich die Gene einer selektierten Zelle anzeigen zu lassen, kann es auch sinnvoll sein, alle Zellen anzuzeigen, in denen dieses Gen aktiv oder inaktiv ist.

Nach der Wahl eines Genes und der Entscheidung, ob dieses Gen aktiv oder inaktiv sein soll, können alle Zellen, für die dieses Kriterium zutrifft, eingefärbt werden. Um die Aktivität des Genes auch bei innen liegenden Zellen zu sehen, müssen die Zellen bei dieser Anzeige ebenfalls transparent sein.

2-D-Darstellung

In der oben beschriebenen 2-D-Darstellung der Zellen ist auch eine Visualisierung des Konzentrationsgradienten einer Substanz möglich. Die Konzentration einer Substanz in einer Zelle wird dazu einer bestimmten Farbe einer Farbskala zugeordnet. Sollen die Konzentrationen einer Substanz in der Zelle dargestellt werden, so werden die Flächen der Zellabbilder entsprechend eingefärbt. Soll das Konzentrationsgefälle einer Substanz im extrazellulären Raum angezeigt werden, so werden die kreisförmigen Zellhüllen in der entsprechenden Farbe dargestellt.

Auch ein Aufzeigen der Zellen, in denen ein bestimmtes Gen aktiv oder inaktiv ist, kann über das Kolorieren der kreisförmigen Zellhüllen realisiert werden.

Die Zellhülle kann in der zweidimensionalen ebenso wie in der dreidimensionalen Darstellung nur eine Farbe gleichzeitig annehmen. Deshalb kann höchstens die Konzentration einer Substanz oder eine beliebige Markierung durch die Einfärbung der Zellhülle dargestellt werden. Zwei Farben könnten zwar als Mischfarbe realisiert werden, der Informationsgehalt würde dabei jedoch verfälscht. Die Mischung erlaubt es nicht mehr, eine optische Zuordnung der Farbe zu den entsprechenden Daten vorzunehmen.

Zelltyp und Zellklasse

Zu den Daten der Zelle, wie sie im Datenmodell definiert wurden, gehört auch ein Zelltyp. Bei der Simulation kann ein Simulator eine Zelle zu einem bestimmten Zeitpunkt als zu einem Zelltyp gehörig klassifizieren. Diese Zelltypen sind Benennungen für Zellen. Sie sind, was die Daten für die Visualisierung betrifft, von dem Zustand der Zelle, also den enthaltenen Substanzen, den extrazellulären Substanzen und den Genen, unabhängig. Alle in einer Simulation vorkommenden Zelltypen bilden eine wohl definierte Menge, die in einer Liste angezeigt werden kann.

Auch bei diesem Attribut können, ähnlich wie bei den Genen, alle Zellen, die dem selektierten Zelltyp entsprechen, markiert werden.

Abgesehen von den vom Simulator vergebenen Zelltypen kann eine Klassifikation der Zellen durch den Betrachter über deren Zustand erfolgen. Dazu können über Regeln benutzerdefinierte Zellklassen, die auf dem Zustand der Zellen basieren, verwendet werden. Der Zustand einer Zelle ist durch die in ihr enthaltenen Substanzen sowie durch die sich im extrazellulären Raum an ihrer Oberfläche befindenden Substanzen und durch die in ihr aktiven bzw. inaktiven Gene definiert. Über einige dieser Merkmale kann eine Regel definiert werden, die eine Zellklasse beschreibt, wobei eine Zelle zur Klasse gehört, wenn sie die Regel erfüllt. Ein Beispiel für eine

Regel könnte folgendermaßen lauten: Die Zellen, in denen Gen A aktiv ist, aber Gen B inaktiv, in denen Substanz K vorhanden, d. h. die Konzentration größer als 0 ist, aber von Substanz L weniger als 2 mol/l enthalten sind und an deren Oberfläche die Substanz X in Konzentration zwischen 1 und 3 mol/l vorkommt, können als Z-Zelle klassifiziert werden, egal wie viele andere Gene oder Substanzen sonst noch vorhanden sind.

Solche benutzerdefinierten Regeln zur Klassifikation von Zellen werden in einer Zellklassen-Liste angegeben. Nach der Wahl einer Klasse können die zu dieser Klasse gehörenden Zellen in der optischen Repräsentation der Zellen durch Einfärbung sichtbar gemacht werden.

Die Visualisierung von Zellklassen kann auch in der zweidimensionalen Darstellung durch Einfärbung erfolgen.

Eine Kombination der Visualisierung des Zelltyps oder einer Zellklasse mit der Darstellung eines Substanzenkonzentrationsgefälles ist ebenfalls denkbar.

Exploration der Daten

Ziel des Visualisierungssystems ist es, die Daten, die von einem Zelldifferenzierungssimulator berechnet werden, darzustellen und damit einem Betrachter die Möglichkeit zu geben, die Resultate der Simulation zu analysieren. Die reine Darstellung aller Daten, vor allem wenn eine große Menge von Daten untersucht werden soll, reicht unter Umständen nicht aus, um Zusammenhänge zu erkennen und Erkenntnisse zu gewinnen. Der Betrachter braucht Mittel, um die Darstellung der Daten zu beeinflussen, damit er die Daten aus verschiedenen Blickwinkeln sehen bzw. einige bestimmte, für ihn interessante Daten fokussieren kann.

In *D-VISION* sollen dem Betrachter mehrere Möglichkeiten der Interaktion mit dem Visualisierungssystem zur Verfügung stehen, die die Form der Darstellung beeinflussen, so dass verschiedene Aspekte der Daten visualisiert werden. Dazu werden die oben beschriebenen Darstellungsmethoden verwendet.

Navigation

Bei der Darstellung der Daten liegt das Hauptaugenmerk auf der dreidimensionalen Visualisierung der Zellen, die als Haufen von (deformierten) Kugeln repräsentiert werden. Der Betrachter muss sich in diesem dreidimensionalen Haufen zurechtfinden, um bestimmte Zellen lokalisieren oder allgemein um sich bestimmte Bereiche näher ansehen zu können.

Hierzu wird eine Abwandlung der in der 3-D-Computergrafik üblichen Virtual-Sphere-Metapher eingesetzt, die Michael Chen 1988 erstmals vorstellte. Dabei ist die Kamera auf einer das zu untersuchende Objekt umgebenden Kugel mit Blickrichtung zum Kugel-Mittelpunkt positioniert und kann auf der Kugel bewegt werden. „Bewegungen der Maus werden als Bewegungen entlang der Längen- und Breitengrade einer virtuellen Kugel interpretiert […]. Diese Metapher ist mittlerweile stark verbreitet, da sie von Benutzern als intuitiv empfunden wird“ [172] und dem menschlichen Verhalten der Untersuchung eines realen Objektes entspricht.

Der Zellhaufen stellt das zu untersuchende Objekt dar. Er kann entsprechend rotiert werden, um um eine Betrachtung von allen Seiten aus zu ermöglichen. Da der Zellhaufen groß sein kann, ist zusätzlich die Möglichkeit, das dargestellte Bild zoomen zu können, von Nutzen, um bestimmte Bereiche näher zu betrachten. Dies kann durch ein „Heranfahren“ oder „Wegfahren“ der Kamera realisiert werden.

Da die Kamera auf den Mittelpunkt des Zellhaufens gerichtet ist, kann beim Zoomen nur der in der Mitte liegende Bereich stark vergrößert werden. Die Randbereiche liegen beim starken Heranzoomen außerhalb des dargestellten Bildes. Entsprechend ist es notwendig, auch eine

Bewegung der Kamera zuzulassen, so dass auch die Ränder des Zellhaufens aus größerer Nähe betrachtet werden können.

Ein Betrachter hat für diese dreidimensionale Navigation fünf Freiheitsgrade (*Degrees Of Freedom, DOF)*, mit denen er das generierte Bild des Zellhaufens beeinflussen kann. Die Navigation kann folgendermaßen beschrieben werden:

- Das zu untersuchende Objekt liegt im Zentrum einer sie umschließenden Kugel (Virtual Shpere).
- Die Größe der Kugel kann variiert werden (Zoom → 1 *DOF*).
- Die Kameraposition entspricht einem beliebigen Punkt auf der Kugeloberfläche (Rotation → *2 DOF*).
- Der Blick der Kamera ist auf den Kugelmittelpunkt gerichtet.
- Außerdem kann die Kamera entlang einer gedachten Ebene, die die Kugel im Punkt der Kameraposition auf der Kugeloberfläche tangiert, verschoben werden. Die Blickrichtung wird dabei beibehalten, so dass der Fokus nicht mehr auf dem Kugelmittelpunkt liegt (Bewegung → 2 *DOF*).

Diese Navigation lässt nur einen Blick von außen auf die Zellen zu. Zwar kann man ganz nah heranzoomen, so dass man die äußersten Zellen mit der Kamera passiert, allerdings werden dann alle Zellen so groß dargestellt, dass man keinen deutlichen Eindruck von der inneren Struktur des Zellhaufens bekommen kann. Eine Möglichkeit, dennoch die inneren Zellen zu sehen, ist es, wie oben schon erwähnt, die Zellen transparent zu machen. Der Betrachter kann entweder alle Zellen oder nur ausgewählte Zellen transparent machen. Ebenso ist es denkbar, dass einige Zellen auch unsichtbar gemacht werden. Werden dann äußere Zellen transparent oder unsichtbar gemacht, so ist ein Blick auf die dahinter liegenden Zellen möglich.

Eine andere Methode, ins Innere des Zellhaufens zu blicken, ist es, ein zweidimensionales Schnittbild zu generieren, denn durch den Schnitt werden alle Zellen in der Schnittebene sichtbar. Das Schnittbild wird durch eine Ebene erzeugt, die senkrecht zur Blickrichtung positioniert ist. Grundlage für die Schnittebene und damit für das Schnittbild ist die davor eingestellte Ansicht auf das dreidimensionale Modell der Zellen. Die Lage der Schnittebene wird also durch die Blickrichtung bestimmt und enthält zu Beginn den Koordinatenursprung. Sie kann aber noch durch Verschiebung entlang der Blickrichtung auf andere Zellen ausgerichtet werden.

Das Schnittbild ähnelt den Bildern, die bei der Lichtmikroskopie von Gewebeproben entstehen. Sind die Proben mehrere Zellschichten dick, so können durch Verlagerung des Fokus verschiedene Schichten untersucht werden. Diese Funktionalität wird durch die Verschiebbarkeit der Schnittebene nachgebildet. Der Betrachter hat die Möglichkeit, den Zellhaufen schichtweise zu durchwandern und sich somit ein Bild des inneren Aufbaus des Zellhaufens zu machen.

Selektion von Zellen

Da die Zustände von Zellen nicht direkt in der dreidimensionalen Darstellung repräsentiert, sondern nur als indirekt raumbezogene Daten in einer separaten Anzeige in Form von Listen angezeigt werden, besteht die Notwendigkeit, Zellen selektieren zu können. Um Zellen selektiv transparent oder unsichtbar machen zu können, ist ebenso eine Selektion notwendig.

Die Auswahl einer Zelle kann durch zwei Methoden realisiert werden. Durch das direkte Anklicken einer Zelle im angezeigten Bild wird diese Zelle selektiert. Allerdings ist diese

Methode, besonders in der dreidimensionalen Ansicht, nicht allzu effektiv, da nur äußere, gerade sichtbare Zellen selektiert werden können. Auch wenn die außen liegenden Zellen transparent sind, können die inneren Zellen nicht angeklickt werden. Das ist nur möglich, wenn die äußeren Zellschichten unsichtbar geschaltet sind. In der zweidimensionalen Ansicht können nur Zellen, die in der Schnittebene liegen, selektiert werden.

Die zweite Methode ergibt sich über eine Auflistung aller Zellen im dargestellten Zellhaufen. Eine Zelle kann durch direkte Auswahl aus der Liste selektiert werden. Zwar kann somit jede auch noch so weit innen liegende Zelle gewählt werden, allerdings ist keine gezielte Selektion aufgrund der geometrischen Position möglich, da die Zuordnung, welche Zelle der Liste welcher Zelle im Bild entspricht, nicht erkennbar ist.

Selektierte Zellen werden durch Einfärbung ihrer Hülle im dargestellten Bild visualisiert.

Wenn eine Zelle selektiert ist, so können ihre Daten, also ihr Zellinhalt, ihre Position und ihre Größe, ihr Zelltyp sowie ihre Zugehörigkeit zu einer Zellklasse, (alphanumerisch) eingesehen werden.

Wenn eine Zelle selektiert ist, ist es möglich, über ihre Entfernung zu anderen Zellen diejenigen zu ermitteln, die als Nachbarzellen in Frage kommen. Mit Hilfe eines entsprechenden Kontrollwerkzeuges, das alle Nachbarzellen anzeigt, können diese dann direkt ausgewählt werden. Dadurch kann der Betrachter von Zelle zu Zelle durch den Zellhaufen navigieren.

Weiterhin ist es sinnvoll, auch mehr als eine Zelle selektieren zu können, um beispielsweise mehrere Zellen transparent oder unsichtbar zu machen. Die Selektion mehrerer Zellen kann wiederum durch die Liste oder durch direktes Anwählen im Bild geschehen.

Eine Inversion der Selektion der Zellen kann auch sehr brauchbar sein, d. h., alle bisher nicht selektierten Zellen werden selektiert und umgekehrt. Beispielsweise können alle Zellen eines bestimmten Zelltyps markiert werden (siehe nächster Abschnitt). Die Auswahl wird invertiert und die nun selektierten Zellen unsichtbar gemacht. Dadurch wird die geometrische Anordnung der Zellen dieses Zelltyps sichtbar.

Eine Selektion mehrerer Zellen ist also sinnvoll, um mit diesen Zellen weitere visuelle Aktionen durchzuführen zu können.

Zellen markieren

Durch die Einfärbung ihrer Hülle in der dreidimensionalen wie in der zweidimensionalen Darstellung können Zellen visuell von anderen Zellen abgehoben und somit markiert werden. Die Markierung und damit gezielte Selektion von Zellen kann nach mehreren Kriterien erfolgen:

- über eine Substanz: Nach der Wahl einer Substanz aus der Liste aller in der Simulation vorkommenden Substanzen und der anschließenden Entscheidung, ob diese Substanz in der Zelle oder im extrazellulären Raum an der Oberfläche der Zelle vorkommen soll, kann eine Bedingung über die Menge formuliert werden. Die Bedingung wird durch ein mathematisches Vergleichssymbol ($<$, $\leq$, $=$, $\geq$, $>$) sowie einen numerischen Wert definiert. Alle Zellen, die den Angaben entsprechen, also beispielsweise Substanz F im Inneren mit einer Menge ≥ 3 enthalten, werden markiert.
- über ein Gen: Die Auswahl eines Genes und die Angabe, ob es in der Zelle aktiv oder inaktiv sein soll, bestimmen das Kriterium, nach dem die Zellen markiert werden.

- über einen Zelltyp oder einer Zellklasse: Alle Zellen eines Zelltyps oder einer Zellklasse können markiert werden. Dazu ist nur eine Auswahl des Zelltyps oder der Zellklasse zu treffen.

Konzentrationen anzeigen

Eine weitere Visualisierungstechnik, mit der die Daten der Zellen genauer in Augenschein genommen werden können, ist die Darstellung des Konzentrationsgradienten einer Substanz. Die Zusammenhänge zwischen Zelltyp oder besser der Differenzierung einer Zelle und ihrem Zellzustand können damit optisch wiedergegeben werden.

In der dreidimensionalen Ansicht sind bis zu drei verschiedene Substanzen wählbar, deren Konzentrationsgefälle über den Zellhaufen hinweg in den Zellen durch TC angezeigt werden. Die TC sind je nach Substanz verschieden eingefärbt. Damit alle TC zu sehen sind, werden alle Zellen transparent dargestellt.

In der zweidimensionalen Ansicht kann das Konzentrationsgefälle von nur einer Substanz über die Einfärbung des Zellinneren realisiert werden.

Die Anzeige der Konzentrationsverteilung im extrazellulären Raum wird über Einfärbung der Hüllen realisiert. Eine gleichzeitige Markierung von Zellen ist entsprechend nicht möglich, da dies ebenfalls über die Einfärbung der Hülle visualisiert wird.

Die Zuordnung der Menge einer Substanz zu einer Farbe wird durch Farbskalen ermöglicht. Dies wird in der dreidimensionalen Ansicht für die Substanzen an der Oberfläche der Zellen und in der zweidimensionalen Ansicht für die Substanz innen und die Substanz außen angewendet. In der zweidimensionalen Ansicht sollten zwei verschiedene Farbskalen verwendet werden, um die Verteilungen besser abschätzen zu können. Wenn allerdings innen und außen die Konzentrationen der gleichen Substanz dargestellt werden sollen, kann auch nur eine Farbskala Verwendung finden. Bei Zellen, die innen und außen die gleiche Konzentration einer Substanz haben, werden die Hülle und das Zellinnere in derselben Farbe dargestellt.

Zeitkomponente

Bisher wurde bei der Beschreibung der Darstellung und der Interaktionsmöglichkeiten des zu konzipierenden Visualisierungssystems der Zeitaspekt der Daten außen vor gelassen. Zelldifferenzierung ist ein zeitlicher Vorgang. Eine Zelle ändert ihren Zustand über die Zeit. Entsprechend werden Zelldifferenzierungssimulationen die Vorgänge in den Zellen über die Zeit nachbilden.

Der zeitliche Aspekt der Daten wird im Datenmodell über ein Zeitattribut der Zustandsdaten der Zellen berücksichtigt. Alle Zustandsdaten mit der gleichen Zeitangabe stellen die Daten des Zellhaufens zu diesem Zeitpunkt dar. Die Zeitangaben in den Daten beziehen sich immer auf einen konkreten Zeitpunkt, d. h., alle Daten, die die gleiche Zeitangabe haben, stellen eine Momentaufnahme des Zustands aller Zellen im Zellhaufen dar. Die gesamten Daten ergeben sich durch aufeinander folgende Momentaufnahmen. Je zeitlich dichter diese Momentaufnahmen sind, desto kontinuierlicher sind die Zustände der Zellen in den Daten wiedergegeben.

Das Visualisierungssystem stellt solche in den Daten gespeicherten Momentaufnahmen der simulierten Zellen optisch dar. Es werden Standbilder eines kontinuierlichen Prozesses visualisiert. In diesen Standbildern sind die oben beschriebenen Methoden der Exploration der Daten möglich. Um nun auch den zeitlichen Aspekt der Daten zu berücksichtigen, ist es nötig, sich von Standbild zu Standbild zu bewegen. Ein solcher Zeitschritt bewirkt die Darstellung der nächsten Momentaufnahme der Daten.

Abgesehen vom natürlichen Zeitablauf können Rückwärtsschritte oder Zeitsprünge gemacht werden. Durch einen Zeit-Schieberegler kann der Betrachter der Visualisierung zu jedem beliebigen Zeitpunkt innerhalb der Simulation springen. Er kann einen Zeitschritt nach vorn oder nach hinten ausführen. Somit kann er den zeitlichen Ablauf des Zustandes der Zellen verfolgen.

Bei Zeitschritten oder Zeitsprüngen bleiben die Darstellungsparameter erhalten. Beispielsweise ist eine selektierte Zelle auch nach einem Zeitschritt selektiert. Ist eine Zelle im nächsten Zeitschritt allerdings nicht mehr vorhanden, weil sie abgestorben ist oder sich geteilt hat, so kann sie natürlich nicht mehr selektiert sein. Liegen hingegen die Informationen vor über die Zelllinien und damit über das Wissen, in welche zwei Zellen sich die selektierte Zelle geteilt hat, so werden bei einer Zellteilung beide Tochterzellen selektiert. Somit kann man nach einem Zeitsprung erkennen, welche Zellen aus einer bestimmten selektierten Zelle hervorgegangen sind.

Wurden z. B. die Zellen einer Zellklasse durch Auswahl dieser Zellklasse markiert, so werden auch beim Ablauf der Zeit zu jedem Zeitpunkt alle Zellen, die zu dieser Klasse gehören, markiert sein.

Wird das Konzentrationsgefälle einer Substanz angezeigt, so kann man die zeitliche Änderung des Gradienten verfolgen.

Weiterhin ist es vorstellbar, die Standbilder zeitlich ablaufen zu lassen. Es entsteht eine Animation. Für eine Wiedergabe der Einzelbilder als Animation ist ein zeitlicher Faktor nötig, der die Geschwindigkeit der Wiedergabe der Einzelbilder festlegt.

Die simulierten Zellvorgänge brauchen auch in der Natur ihre Zeit. Gemessen wird üblicherweise die Dauer eines Zellzyklus, also die Zeit, die zwischen zwei Zellteilungen vergeht. Der Zellzyklus ist unterschiedlich lang. Er kann bei Leberzellen eines ausgewachsenen Säugers über ein Jahr dauern. In der frühen Embryogenese dauert der Zellzyklus je nach Spezies zwischen acht und 60 Minuten [170]. Ein Simulationsmodell sollte entsprechend auch diesen zeitlichen Ablauf nachbilden. Zwischen zwei vom Simulator gelieferten Momentaufnahmen liegt ein ganz bestimmter Zeitraum. Werden die Standbilder dieser Momentaufnahmen in genau diesem zeitlichen Abstand angezeigt, so wird die Visualisierung in Echtzeit erfolgen. Da die Vorgänge etliche Zellzyklen dauern, ist eine schnellere Wiedergabe im Zeitraffer sinnvoll.

Ein begrenzender Faktor für die Wiedergabe einer Animation ist die Leistungsfähigkeit des Rechners, der die Visualisierung darstellt. Da für jedes Einzelbild etliche Berechnungen für die dreidimensionale Darstellung vorgenommen werden müssen, kann die Wiedergabe von Einzelbildern sehr stockend sein.

Das Datenmodell ermöglicht es, dass die zeitlichen Abstände zwischen den Momentaufnahmen unterschiedlich groß sind. Diese Eigenschaft erschwert jedoch eine flüssige Wiedergabe der Animation.

Eine Änderung der Darstellungsparameter während der Wiedergabe ist zwar denkbar, kann aber unter Umständen zu Fehlinterpretationen der dargestellten Daten durch den Betrachter führen.

Neben der direkten Navigation durch die Zeit mit Hilfe eines Zeit-Schiebereglers ist auch die Navigation mit Hilfe einer selektierten Zelle möglich. Mittels eines entsprechenden Navigationselementes kann zur Mutterzelle oder zu einer der Tochterzellen gesprungen werden. Beim Sprung zurück zur Mutterzelle wird zu dem Zeitpunkt gesprungen, an dem die Mutterzelle sich gerade noch nicht geteilt hat. Die Auswahl einer der Tochterzellen verursacht einen Sprung vorwärts in der Zeit zu dem Zeitpunkt, an dem sich die zuvor selektierte Zelle

eben geteilt hat. Somit ist es möglich, sich zeitlich durch den Zellhaufen entlang der Zelllinien zu bewegen. Voraussetzung dafür ist natürlich, dass die entsprechenden Daten vom Simulator geliefert werden.

Definition von Zellklassen im Kontext D-VISION

Da *D-VISION* die Visualisierung von Zelldifferenzierung zum Ziel hat, müssen neben der Darstellung der Zellen vor allem die verschiedenartigen Zellen hervorgehoben werden. Abgesehen vom Zelltyp, einem im Datenmodell für jede Zelle enthaltenen Attribut, das vom Simulator vergeben wird, stehen für das Kennzeichnen verschiedener Zellen die Daten zur Verfügung, die den Zustand der Zelle repräsentieren. Der Zustand der Zellen wird von drei verschiedenen Merkmalsarten bestimmt, nämlich den Substanzen in der Zelle, den Substanzen im extrazellulären Raum, die sich an der Oberfläche der Zellen befinden, und den Genen in der Zelle. Jedes Merkmal, also die Konzentration einer Substanz in oder an der Zelle oder die Aktivität eines Genes, ist Teil des Zustandes der Zelle. Allerdings wird es einige Merkmale geben, die insbesondere für eine bestimmte Funktionalität der Zelle zuständig sind und damit die Differenzierung dieser Zelle mit verantworten. Es ist aber davon auszugehen, dass nicht nur ein Merkmal die Differenzierung einer Zelle auslöst. Vielmehr sind es Kombinationen von mehreren Merkmalen, die die Gesamtfunktionalität der Zelle ausmachen. Um dem gerecht zu werden, enthält das Visualisierungssystem das Konzept der Zellklassifikation.

Zellklassen werden durch Regeln definiert, die die Merkmale von Zellen betreffen. Die verwendeten Regeln sind boolesche Ausdrücke und aus einzelnen booleschen Argumenten aufgebaut. Die Argumente werden mit logischen Verknüpfungen wie *UND* und *ODER* sowie Klammern zu einer Regel zusammengefügt. Außerdem kann der boolesche Wert eines Argumentes durch Negation invertiert werden. Als Argumente werden in ihrer Aktivität bzw. in ihrer Menge exakt spezifizierte Gene und Substanzen benutzt. Das boolesche Argument für ein Gen überprüft, ob dieses Gen aktiv oder inaktiv ist. Durch einen mathematischen Vergleich mit einem Zahlenwert ($<$, $\leq$, $=$, $\geq$, $>$) wird ein boolesches Argument für Substanzen formuliert. Erfüllen die Merkmale einer Zelle die Regel einer Klasse, so gehört sie zu dieser Klasse.

Das Konzept der Zellklassen ist ein mächtiges Werkzeug in der Hand des Benutzers des Visualisierungssystems. Mit Hilfe einer entsprechend umfangreichen Regel können Zellen klassifiziert und lokalisiert werden, die sich zu einem bestimmten Typ differenziert haben. Genauso können durch eine einfache Regel Zellen mit einem bestimmten Merkmal gefunden werden. Die Zellklassen dienen der Exploration und Analyse der Daten.

Die Zellklassen sind Bestandteil des Visualisierungssystems und damit unabhängig vom Simulator. Zellklassen werden nicht vom Simulator vorgegeben, sondern vom Benutzer des Visualisierungssystems erstellt. Um sie nicht bei jeder Benutzung neu erstellen zu müssen, ist es möglich, die Klassendefinitionen abzuspeichern. Dabei sind drei verschiedene Arten von Klassendefinitionen möglich. Globale Klassendefinitionen sind in allen Simulationen verwendbar. Lokale Klassendefinitionen sind nur für die aktuelle Simulation, allerdings auch in späteren Aufrufen, verfügbar. Temporäre Klassendefinitionen können nur in der aktuellen Simulation und auch nur bis zum Laden einer anderen Simulation oder dem Beenden des Visualisierungssystems benutzt werden. Entsprechend müssen die globalen und die lokalen Klassendefinitionen persistent gemacht werden. Sie werden in der Datenbank des Aufbereiters gespeichert. Folgendes Verhalten ergibt sich daraus: Beim Öffnen des Visualisierungssystems werden die globalen Klassendefinitionen eingelesen. Wird eine Simulation geladen, werden zusätzlich die entsprechenden lokalen Klassendefinitionen der Datenbank entnommen. Sie stehen somit dem Benutzer zur Verfügung. Abgesehen vom Erstellen neuer Klassendefinitionen können vorhandene eingesehen, verändert und gelöscht werden. Bei der Verwendung eines zentralen Aufbereiters, auf den mehrere Instanzen des Visualisierungssystems zugreifen, sollte für das Editieren und Löschen von Klassendefinitionen eine Rechteverwaltung implementiert

werden, damit nicht die abgespeicherten Klassen anderer Benutzer willkürlich verändert oder gelöscht werden können.

5.3. Realisierung des Konzepts in anderen Anwendungsbereichen

Im Folgenden werden Realisierungen und Projekte vorgestellt, die die hier für die computergestützte Biochemie beschriebenen Konzepte für weitere Bereiche interpretieren. Notwendigerweise werden dabei jeweils nur Teile des Konzepts umgesetzt. Die beiden ersten Projekte gehen zeitlich der in diesem Text vorgestellten Hauptarbeit im Bereich computergestützte Biochemie voraus und lieferten wertvolle Erkenntnisse, die in das spätere Konzept einflossen.

5.3.1 Gläserner Reaktor

Im Projekt „Gläserner Reaktor“ wurde in den Jahren 1998 bis 2001 eine interaktive Steuerungs- und Visualisierungskomponente verbunden mit einem existierenden Simulator für die im Kapitel „Kontrolle chemischer Prozesse“ der Anforderungsanalyse beschriebenen Ziegler-Synthese für hochdichtes Polyethylen (HDPE).

Das Reaktionsmodell wurde spezifisch für diesen Prozess erstellt von der Hoechst Prozessleittechnik, später Hoechst Research and Technologies GmbH, später Aventis Research, später Axiva GmbH, jetzt Siemens Axiva GmbH in Zusammenarbeit mit der Hostalen GmbH, jetzt Basell Polyolefins Company N.V. [173]. Als Simulatorwerkzeug wurde der Differenzialgleichungslöser von AspenTech [55] verwendet. Diese Modell-Simulatorkombination wurde bis dahin bereits zur (prozessbegleitenden) Prozesskontrolle verwendet.

Neben dem prozessbegleitenden Einsatz sollten nun für das im Prozessmodell gebundene Know-how neue Anwendungsfelder erschlossen werden. Konkret wurden zwei Ziele verfolgt und erreicht:

1. Einsatz des Know-hows als marketingunterstützendes Element einer Messepräsentation
2. Nutzbarmachen des Werkzeugs „Simulation“ als Mittel zur Unterrichtung von „interessierten Laien“, d. h. von Anwendern mit (chemischem) Prozess-Know-how, aber ohne Vorkenntnisse in der Benutzung von Simulationswerkzeugen.

Beide Aufgaben wurden über ein flexibles System gelöst, bestehend aus drei Komponenten (siehe Abbildung 120):

1. einer Präsentationsumgebung, bestehend aus einem VRML-kompatiblen [98] 3-D-Visualisierungswerkzeug, dem CasusPresenter [174] und einer VRML-Szene (VRML beinhaltet die Möglichkeit, neben 3-D-Geometrie und Aussehensbeschreibungen auch Programmcode in einer VRML-Szene zu verwalten. Für diese Anwendung wurde stark auf dieses Feature zurückgegriffen.)
2. einem in der Programmiersprache FORTRAN programmierten Simulator der Firma [55]
3. einem Kopplungsagenten, der die Eingaben des Nutzers interpretiert und für den Simulator aufbereitet, der die Verbindung zwischen Präsentationsumgebung und Simulator etabliert und aufrechterhält und schließlich die Ergebnisse des Simulators an die Präsentationsumgebung weiterreicht.

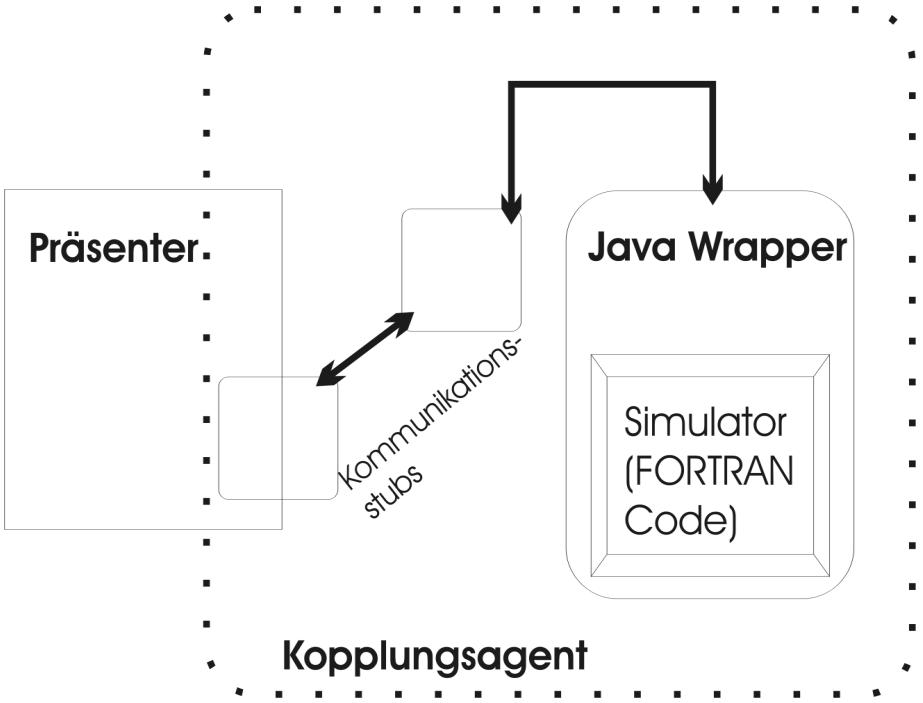


Abbildung 120 Komponenten Gläserner Reaktor

Die Präsentation zeigt den chemischen Reaktor mit seinen Zuflüssen und dem Abfluss in drei verschiedenen Modi:

- Geschlossen: ähnlich dem Aussehen des realen Reaktors
- Geöffnet: Der Blick auf Rührwerk und Reaktionsraum ist sichtbar (siehe Abbildung 121).
- Microphase: Die wachsenden Polymerknäuel sind im Detail sichtbar (siehe Abbildung 122).

In einem zusätzlichen Modus werden die verschiedenen Modi sowie Grundlagen des chemischen Verfahrens in einer filmartigen Sequenz, der sogenannten Autoshow, demonstriert.

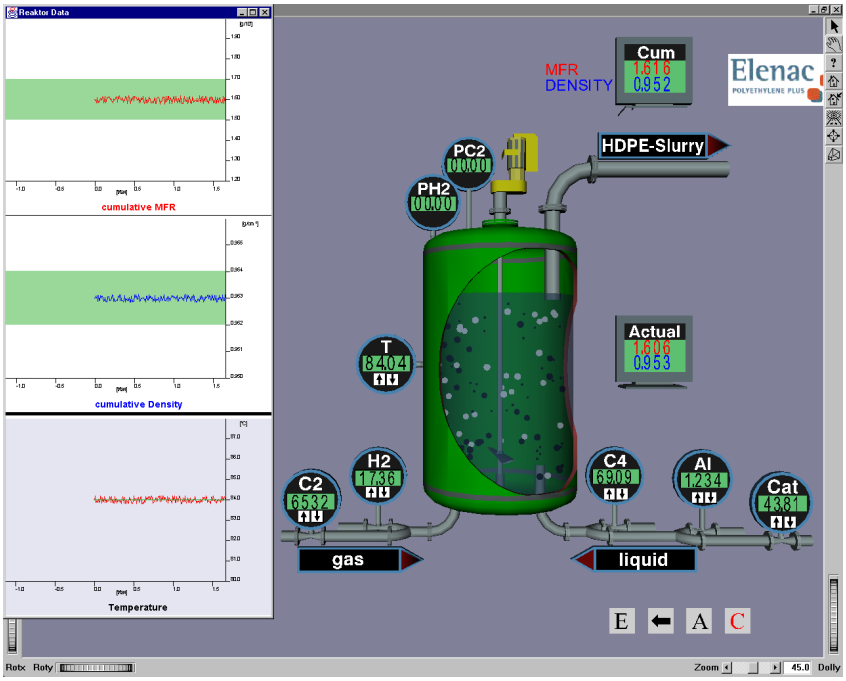


Abbildung 121 Gläserner Reaktor offen

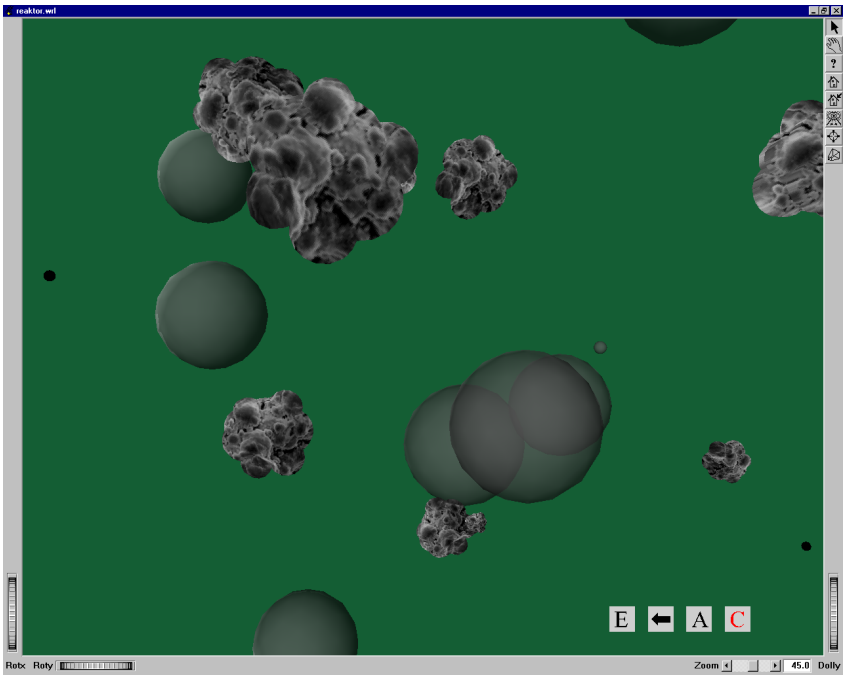


Abbildung 122 Microphase

Das eingesetzte Visualisierungswerkzeug, CasusPresenter, ermöglicht sowohl die Präsentation als Stereoprojektion im Messekontext als auch die Präsentation auf einem normalen Laptop (Schulung und Unterrichtung).

Der Kopplungsagent hat zwei wesentliche Aufgaben: Zum einen war es nötig, eine Brücke zwischen Simulator und Präsentation zu schlagen. Dies ist insbesondere deswegen problematisch, da zwei verschiedene Laufzeitumgebungen für diese beiden Anwendungsteile zum Einsatz kommen mussten. Durch die Anforderungen der Präsentation (sowohl 3-D-Stereo als auch konventioneller Laptop) und die darauf bauende Auswahl der VRML als computergraphische Technologie stand als vollwertige Laufzeitumgebung nur Java [43] zur Verfügung. Der Simulator dagegen ist auf ein Fortran-Laufzeitsystem angewiesen.

Zum anderen ist der Kopplungsagent dafür verantwortlich, die Benutzereingaben für den Simulator umzusetzen und dessen Ergebnisse weiterzuleiten. Zu dieser Umsetzung gehört insbesondere die Implementierung des Zeitverhaltens der virtuellen „Stellelemente", d. h. der Ventile. Auf der Ausgabeseite rechnet der Kopplungsagent die vom Simulator erzeugten Werte um, um wiederum das korrekte Zeitverhalten zu erhalten. Hierzu kommt auch, dass diese Daten gespeichert werden müssen, um im Rahmen der Präsentation einen zeitlichen Verlauf auswerten zu können, den der Simulator so nicht liefern kann.

5.3.2 ETOILE

Das Projekt ETOILE, kurz für *Environment for Team, Organisational and Invidual Learning in Emergencies*, wurde von der Europäischen Gemeinschaft unter dem IST-Programm (IST29086) gefördert und von 1998 bis 2001 bearbeitet. In der Anforderungsanalyse wurden bereits die grundlegenden Probleme des Trainings für Notfallsituationen besprochen. Anders als beim (individuellen) Training für Rettungspersonal im weiteren Sinne gehören zur Zielgruppe dieses Projekts Personal, Management und Ausführende, deren überwiegende Arbeitsaufgabe die normale Durchführung, Überwachung und Steuerung ziviler Betriebsabläufe ist. Notfälle treten hierbei nur sporadisch auf, so dass korrekte Reaktionen auf diese Fälle in der normalen Arbeitsroutine nicht erlern- bzw. trainierbar sind.

Als Modellszenarien wurden in diesem Projekt zwei Anwendungsfälle bearbeitet:

- Notfall in einem Kernkraftwerk (Projektpartner Technatom, S. A., Spanien und Iberdrola, S. A., Spanien)
- Brand eines U-Bahn-Wagens im Tunnel (Projektpartner Labein, Spanien und die Metro von Bilbao, Spanien).

Für die pädagogisch angemessene Umsetzung der Trainingssysteme stand das *Centre for Studies in Advanced Learning Technology* der Universität von Lancaster in Großbritannien zur Verfügung. Die computertechnische Entwicklung wurde vor allem von STN Atlas Elektronik und dem Fraunhofer Anwendungszentrum für Computergraphik in Chemie und Pharmazie übernommen.

Entsprechend der Anforderungsanalyse sind beim Training für Notfallsituation zwei Hauptprobleme zu überwinden: Zum einen muss eine Möglichkeit geschaffen werden, eine große und gegebenenfalls heterogene Anzahl von Trainierenden für das Training zu versammeln; zum anderen muss ein geeigneter Trainingsort gefunden werden, im Falle eines Kernkraftwerkes wäre dies das gesamte Kraftwerk samt zugehöriger Umgebung.

Die grundsätzliche Lösungsidee des Projektes ist die Verlagerung des Trainingsorts in die virtuelle Realität (VR) und das schrittweise Ersetzen von „Mitspielern" durch Intelligente Software-Agenten (siehe entsprechendes Grundlagenkapitel). Das wichtigste Trainingsziel unter den gegebenen Umständen ist nicht die tatsächliche Bekämpfung des Notfalls am Gefahrenherd, etwa einem Brand, es besteht vielmehr in der Koordination verschiedener Einsatzkräfte, der Kommunikation mit verschiedenen Stellen und Organisationen etc.

Schrittweises Ersetzen von Trainierenden heißt hierbei, dass zunächst operative Einsatzkräfte durch Agenten ersetzt werden. Weitere Trainierende können sukzessive durch Agenten ersetzt werden, falls sie aus trainingskoordinatorischen Gründen nicht am Training teilnehmen können. Neben Personen werden auch weitere Simulatoren über die Agentenschnittstelle angekoppelt. So wurde etwa für die Kraftwerkssimulation ein Simulator für die Berechnung der Windgeschwindigkeit und die Ausbreitung radioaktiver Stoffe angebunden.

Folglich besteht das ETOILE-System aus zwei Komponenten: einer Desktop-VR-Anwendung, um das virtuelle Trainingsszenario darzustellen, und einer Agentenkomponente. Verbunden werden diese Komponenten durch eine Kommunikationsinfrastruktur (siehe Abbildung 123).

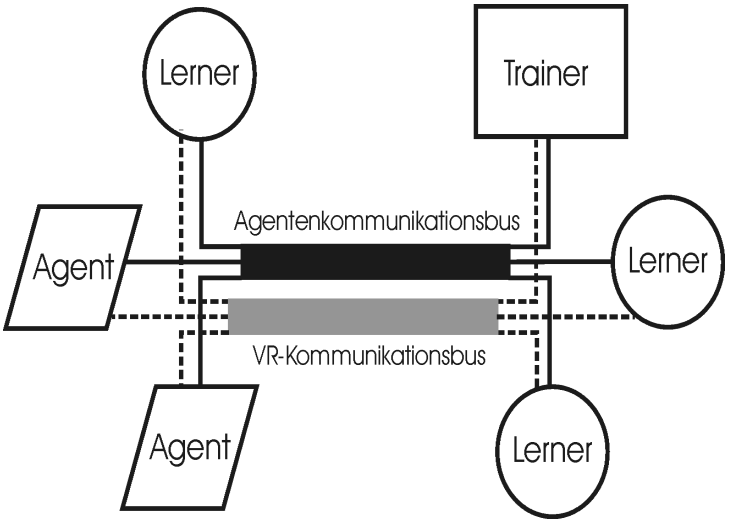


Abbildung 123 Kommunikationsinfrastruktur ETOILE

Für diese Arbeit sind vor allem die Komponente der Agenten und der Kommunikation von Interesse.

Die Trainingsapplikation wurde im Bereich Visualisierung per C++ und der 3-D-Bibliothek OpenGVS [175] erstellt; der Bereich der Kommunikation, der Benutzungsschnittstelle und der Intelligenten Agenten wurde in der Programmiersprache Java implementiert.

Kern der Implementierung der Intelligenten Agenten ist die *Rules Engine*. Das Verhalten eines ETOILE-Agenten wird durch so genannte *Produktionsregeln* definiert. Dabei wurde auf die Rule-Engine von Neuron Data (später Blaze), den *Advisor*, zurückgegriffen [176]. Diese einfach zu handhabenden *if-then-else*-Regeln (siehe Abbildung 124) haben sich als gut geeignet erwiesen, komplexes Verhalten zu implementieren. Dies gelingt sogar den Fachexperten der Anwendungspartner – ein wichtiges Ziel der ETOILE-Entwicklung, bedenkt man, dass Experten für Agentensysteme nicht notwendigerweise Experten für das richtige Verhalten bei Notfällen in Kernkraftwerken sind.

```
rule rule_2 is

if AgentState.pushbutton_task = true and
AgentState.currentLocation.firstString() <> "C"

then {
```

```
select
AgentState.agent.getDistribution("rule_2").chooseAction(AgentState.agent
.currentTimestamp())
case 0 :
       AgentState.moveaction = AgentState.agent.action(
       EtLog.ETLOG_EXTERNAL_ACTION,"MOVE_TO_C",
       10,
       "rule_2",
       AgentState.agent.currentTimestamp(),
       0);
}.
```

Abbildung 124 Regeln

Ein Produktionsregelsystem allein ist ausreichend, um reaktives Verhalten eines Agenten zu modellieren, etwa in der Art: *Wenn* es brennt, *dann* löse Alarm aus. Im vorliegenden Fall soll allerdings menschliches Verhalten glaubhaft nachgebildet werden. In diesem Fall ist reaktives Verhalten, also Verhalten, das durch externe Ereignisse ausgelöst wird, allein nicht ausreichend. Um nun auch proaktives Verhalten modellieren zu können, wurde das Produktionsregelsystem erweitert. Eingefügt wurde ein zeitabhängiges Konstrukt, um Regeln nach einem Zeitablauf ohne externes Ereignis auslösen zu können. Zusätzlich wurde ein stochastisches Element eingefügt, das es erlaubt, verschiedene Reaktionsvarianten auf ein Ereignis hin zu implementieren.

Die Kommunikationskomponente jedes Elementes, sei es Agent oder Trainierender, vereinheitlicht die Systemsicht für alle Beteiligten. Es ist wichtig zu bemerken, dass in diesem System nicht nur Agenten miteinander kommunizieren, vielmehr ist die Kommunikationsinfrastruktur genauso für die Kommunikation der menschlichen „Mitspieler" verantwortlich. Aus diesem Grund war es nicht möglich, eine Standard-Agentenkommunikationssprache zu verwenden, da diese sich nicht transparent für menschliche Kommunikation einsetzen lässt.

Als Lösung wurde ein System, bestehend aus *Kommunikationsmitteln* und *Vokabularen*, entwickelt. *Kommunikationsmittel* sind dabei auch real existierende Mittel, wie Telefon oder Funk. Die benutzen *Performatives* haben also den Typ *TELEFONMESSAGE* etc. Der Inhalt der Kommunikation wird über *Vokabulare* geregelt. Das *Active Vocabulary* ist dabei das Vokabular, das von einem Kommunikationsteilnehmer aktiv benutzt, also gesendet werden kann. Das *Passive Vocabulary* ist entsprechend das Vokabular, das verstanden wird.

Die folgenden Abbildungen (Abbildung 125) stellen einige Szenen aus verschiedenen Trainingsszenarien dar.

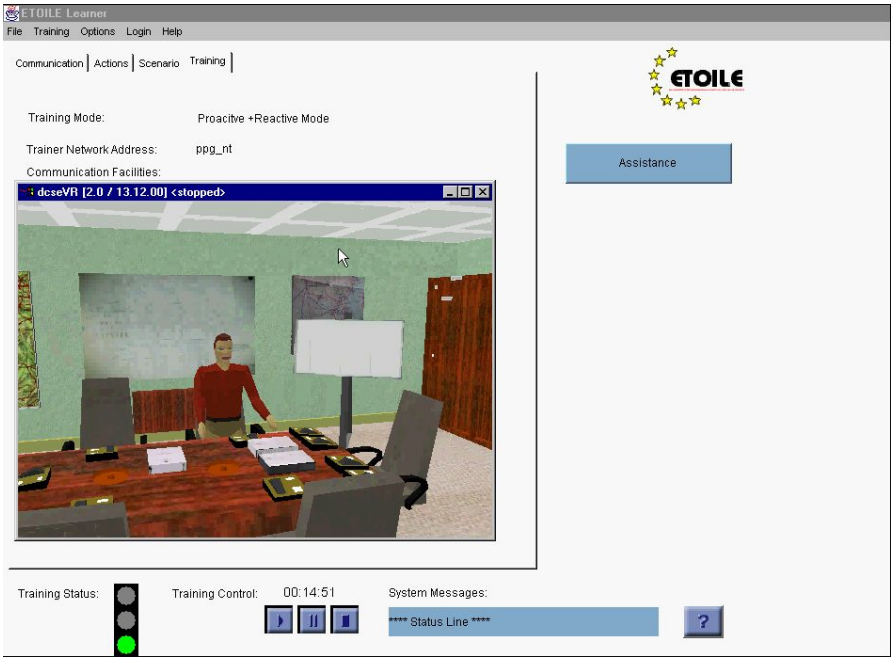


Abbildung 125 ETOILE-Screenshots

5.3.3 SpacemantiX

Das von der EU geförderte Projekt „SpacemantiX“ (IST-2001-34159) mit einer Laufzeit von 36 Monaten seit Mai 2002 beschäftigt sich mit dem „Kombinieren von räumlicher und

semantischer Information in Produktdaten“. Ausgangspunkt ist die Erkenntnis, dass dreidimensionale Modelle in kommerziellen Anwendungen außerhalb von Planungs- und Entwicklungsabteilungen trotz weiter Verfügbarkeit von CAD- und 3-D-Modellierungswerkzeugen wenig genutzt werden. Das Projekt „SpacemantiX hat sich zur Aufgabe gesetzt, die Verwendbarkeit von 3-D-Modellen zu erweitern und sie z. B. in Bereichen wie Produktkatalogen, Montage-, Bedienungs- und Wartungsanleitungen sowie 3-D-Planungswerkzeugen einsetzbar zu machen.

Die Lösungsstrategie des Projektes ist zweifach und basiert auf der Anreicherung der 3-D-Modelle durch semantische Informationen. Diese Informationen werden im zweiten Schritt durch Anwendungen genutzt, um neue Funktionalitäten zur Verfügung zu stellen.

Semantische Informationen reichen dabei von Angaben zur Branche, etwa „Möbel“, „Automobilbau“ etc., über Informationen zur Verwendungsart des beschriebenen Objekts, etwa „Schreibtisch“, bis zu Hinweisen zum Verhalten des Objekts, wie z. B. Verhältnis zu anderen Objekten einer Szene, „bevorzugter Standort: Fußboden“, „vor dem Objekt darf im Abstand x kein anderes Objekt dauerhaft platziert werden“. Wichtig sind auch Informationen wie: „kann mit Objekt X kombiniert werden“. Diese sind kontextabhängig und können vom Objekt dynamisch geändert werden.

Als exemplarische Anwendungsbereiche wendet sich SpacemantiX an die Bereiche

- Möbelhandel und Architektur
- Maschinenbau und Automobildesign
- Spielzeugbranche.

Eine Anwendung, die auf die Objektkomponenten dieses Projektes aufbaut, kann die semantischen und verhaltensbeschreibenden Informationen dann zur aktiven oder passiven Assistenz einsetzen. Aktiv heißt, das Programm schlägt von sich aus sinnvolle Aktionen vor; passiv bedeutet, das Programm prüft auf Wunsch die Einhaltung der Planungsregeln.

Im Rahmen des Projektes wurden und werden an der Johann-Wolfgang-Goethe-Universität Franfurt am Main Anwendungen entwickelt, die im Bereich der Möbelplanung die im Projekt entwickelten semantischen Informationen auswerten. Nach dem hier vorgestellten Konzept wird jedes Objekt durch eine Entität repräsentiert, die ihr Verhalten durch die Anbindung an Simulatoren (im Projektkontext *Funktionen* genannt) steuert. Die Abbildung 126 zeigt verschiedene an der JWG-Universität entwickelte SpacemantiX-Anwendungen.

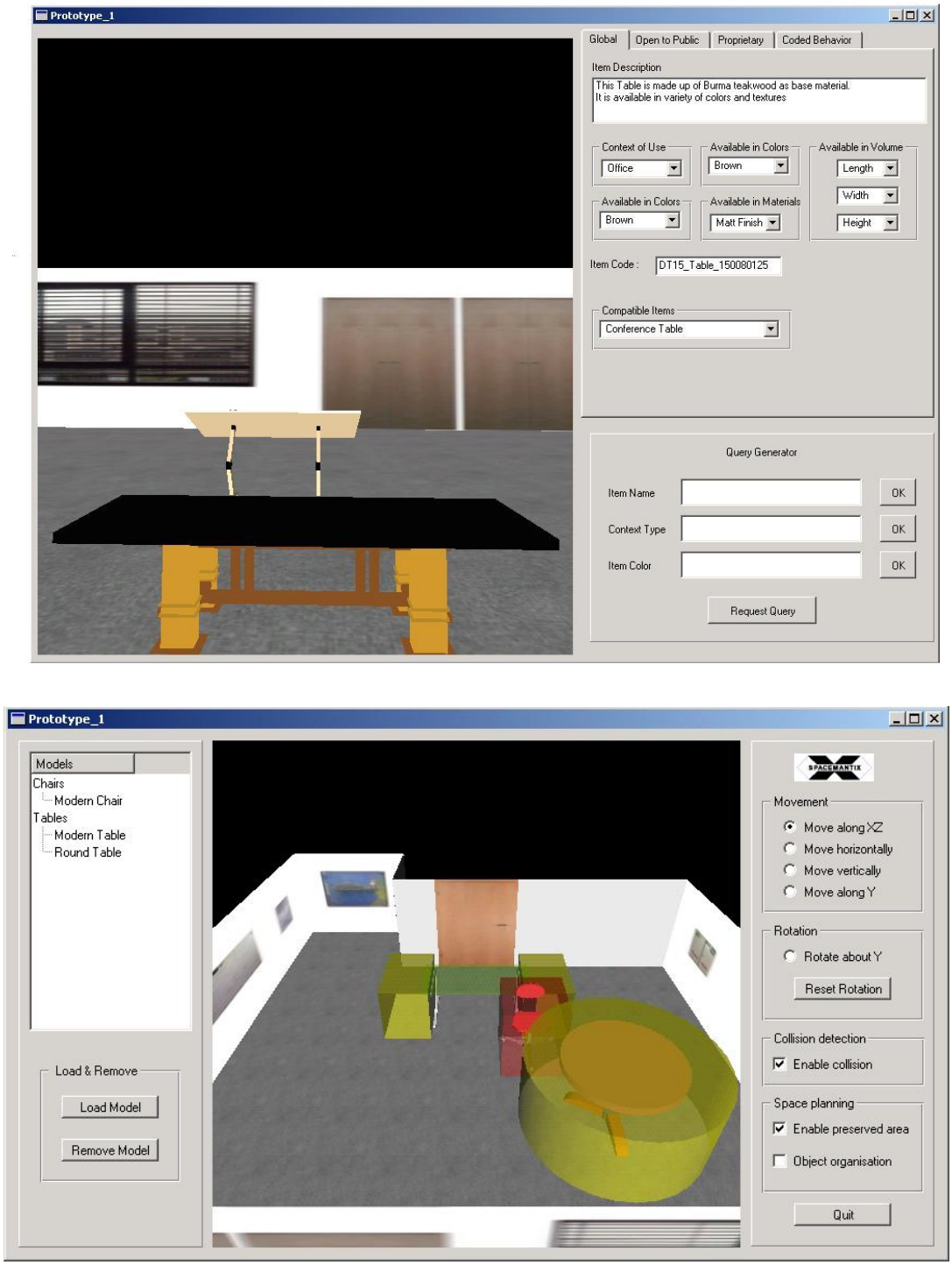


Abbildung 126 Spacemantix-Screenshots

5.4. Zusammenfassung

In diesem Kapitel wurde die Umsetzung des vorgestellten Konzepts in einer Reihe von Anwendungen vorgestellt. Dabei wurden alle drei besonders beachteten Bereiche der computergestützten Biochemie betrachtet, nämlich Molecular Modeling, Systembiologie und Zelldifferenzierung.

Zunächst wurde jedoch ein neues Ein-/ Ausgabegerät passend zum Interaktionskonzept vorgestellt, die *Virtual Glove Box*. Ein hierzu passendes User-Interface-System mit haptischer Unterstützung vervollständigt die Interaktionsimplementierung.

Anschließend wurden für alle drei Bereiche jeweils dedizierte Implementierungen vorgestellt. Im Bereich *Systembiologie* wurde ein Simulationsexperiment exemplarisch beschrieben. Das Kapitel endet mit erweiterten Umsetzungen des Konzepts für Bereiche jenseits der computergestützten Biochemie; es wurden exemplarisch Projekte aus den Bereichen „Visualisierung der Prozesskontrolle chemischer Prozesse“, „Training in Notfallsituationen“ und „Einrichtungsplanung“ vorgestellt.

6. Ergebnisse

6.1. Übersicht

Die Betrachtung der vorgestellten Grundlagen und die anschließende Anforderungsanalyse ergaben spezifische Anforderung an die Simulation und die Visualisierung im Kontext der computergestützten Biochemie. Als besondere, bis dato nicht oder nicht befriedigend adressierte Anforderung wurde die dreidimensionale Natur der betrachteten Anwendungsfälle identifiziert. Wichtigste Erkenntnis war dabei, dass es eines *Gesamtkonzeptes* bedarf, das die beiden Felder der Visualisierung und der Simulation im Hinblick auf diesen Anwendungsfall geeignet miteinander verbindet.

In dieser Arbeit wurde ein solches Gesamtkonzept erarbeitet und mit Hilfe verschiedener konkreter Realisierungen validiert. Dieses Gesamtkonzept besteht aus zwei aufeinander abgestimmten Teilkonzepten – einem Simulationskonzept und einem Visualisierungskonzept.

Als Basis für das Gesamtkonzept wurden in beiden Teilgebieten eigenständige *Entitäten* identifiziert, die miteinander kommunizieren und Informationen austauschen. Im Bereich der Simulation der computergestützten Biochemie betreffen diese kombinierbaren „Bausteine“ einer Simulationsstudie sowohl die Verfahren der Simulationstechnik als auch die Modellierung des Simulationsgegenstandes. So kann etwa ein deterministisches Verfahren mit einem Baustein eines stochastischen Verfahrens gekoppelt werden. Ebenso kann zum Beispiel der Modellgegenstand „Zelle“ erstmals effektiv aus räumlichen „Bausteinen“ aufgebaut werden. Diese Bausteine können z. B. das Simulationsvolumen überdecken oder aus „Bausteinen“ der Zelle, etwa verschiedener Organellen, bestehen.

Für den Bereich der Visualisierung und Interaktion lassen sich ebensolche *Entitäten* identifizieren, diesmal in Form von Visualisierungskomponenten. Solche Visualisierungskomponenten wurden in dieser Arbeit genauer untersucht und klassifiziert. Wichtig ist in dieser Hinsicht, dass die beiden Teilkonzepte nicht nur von der Struktur her kompatibel sind, sondern gemeinsam zu einer neuen, interaktiven Simulationsmethodik führen.

Das verbindende Element ist die neuartige Konstruktion des *Kopplungsagenten*, der sowohl die Simulationsbausteine kapselt als auch die Visualisierungs- und Kommunikationsinfrastruktur aufbaut. Abbildung 63 Simulation Agenten-Kopplungillustriert dies. Der gesamte Versuchsaufbau der Simulationsstudie besteht so aus einer Kopplung *heterogener* Agenten. Dies bedeutet, dass nicht *innerhalb* eines Agenten heterogene Technologien vereinigt werden sollen. Es werden vielmehr *verschiedenartige* Agenten zu einem komplexen System zusammengestellt.

Somit bietet das hier vorgestellte Konzept erstmals einen flexiblen Ansatz, *dreidimensionale* Phänomene angemessen zu modellieren. *Modellierung* bezieht sich dabei sowohl auf die angemessene Visualisierung als auch auf das Simulationsmodell.

Es wurde beschrieben, wie ein *Kopplungsagent* durch zwei Schnittstellen definiert wird: eine Schnittstelle nach „innen“ zur Simulation sowie eine Schnittstelle nach „außen“ zu den anderen Agenten.

Die Schnittstelle nach „innen“ ist verantwortlich für

- die Konfiguration der Randbedingungen der Simulation
- die Steuerung des Simulationslaufs
- die Visualisierung der Ergebnisse.

Die Schnittstelle nach „außen“ ist für die Kommunikation zwischen den Agenten verantwortlich.

Das hier vorgestellte Konzept ist unabhängig von einer speziellen Agentenplattform, denn es ergibt keinen Sinn, im Kontext eines komponentenbasierten Simulations- und Interaktionskonzeptes für die computergestützte Biochemie eine weitere Agentenimplementierung der bereits unüberschaubaren Zahl vorhandener generischer Agentensysteme hinzuzufügen. Vielmehr wurde dargestellt, unter welchen Voraussetzungen *beliebige* Agentensysteme für den Einsatz als Kopplungsagent genutzt werden können.

Hierzu wurde ein eigener „Dialekt“ der Standard-Agenten-Kommunikationssprache der FIPA in Form einer Auswahl der benötigten *Performatives* erstellt, die *BioSLang*. Dazu passend wurde eine an die Bedürfnisse der computergestützten Biochemie angepasste Inhaltsbeschreibungssprache definiert, die *BiSCeD*.

Die Idee des Kopplungsagenten wird für die Visualisierung und Interaktion, wie oben beschrieben, als Visualisierungskomponente interpretiert. Visualisierungskomponenten werden in diesem Kontext vereinfacht als eine Softwareeinheit verstanden, die ihre Repräsentation unter eigener Verantwortung innerhalb eines Frameworks kontrolliert.

In dieser Arbeit wurden die Rolle und die Einordnung der Ein-/ Ausgabe innerhalb eines Visualisierungsagenten ausführlich erörtert.

Das vorgestellte Interaktionskonzept sieht vor, dass der Benutzer direkt mit der Visualisierung interagieren kann. Durch die enge Verzahnung des Interaktions- und Visualisierungskonzepts mit dem Simulationskonzept ist es möglich, nun auch direkt mit der Simulation zur Laufzeit zu interagieren.

Zur Betonung der verbesserten interaktiven Möglichkeiten einer auf Kopplungsagenten basierenden Simulationsstudie wurde ein neues, multimodales Ein-/ Ausgabegerät vorgeschlagen, die *Virtual Glove Box*. Dieses Gerät fügt einem 3-D-Präsentationssystem eine haptische Komponente zu. Diese haptische Komponente setzt sich auf der Seite der Software als Teil des Komponentenkonzepts zur Visualisierung fort.

Das Konzept der *Virtual Glove Box* besteht aus drei Teilen:

1. einem Konzept für Eingabegeräte
2. einem Konzept für Ausgabegeräte
3. einer gemeinsame Metapher, die die Geräte logisch in den Anwendungskontext einbettet.

Das Konzept für die Eingabegeräte beschreibt die direkte und manipulative Eingriffsmöglichkeit des Benutzers in Simulationsstudien. Dazu passend wurde ein visuelles Präsentationssystem, basierend auf einem Stereoprojektorsystem, für die 3-D-Ausgabe dargestellt.

Das Prinzip einer (realen) Glove Box ist es, den Benutzer vom Arbeitsbereich zu trennen und ihm dennoch die Möglichkeit zu geben, die zu bearbeitenden Substanzen direkt zu manipulieren. Eine solche Trennung von Benutzer und Arbeitsbereich ist immer dann nötig, wenn eine direkte Handhabung der Substanzen und Gegenstände für den Benutzer zu gefährlich ist (Substanzen sind giftig, es treten Gase aus etc.) oder wenn das Arbeitsmaterial vom Benutzer oder der Umgebung kontaminiert werden würde (Arbeit in geschlossener oder abgeschlossener Atmosphäre). Der Aufbau eines solchen Apparates besteht aus einer durchsichtigen Kammer, in deren Seitenwand Öffnungen vorgesehen sind. Diese Öffnungen

dienen entweder der Beladung (sie sind für die Umsetzung als VR-System nicht relevant) oder sind Einlässe in Handschuhe. Durch diese Handschuhe greift der Benutzer in die Kammer, ohne mit dem Inhalt direkt in Kontakt zu kommen (siehe Abbildung 68).

Diese Metapher ist aus dem Anwendungsbereich der Biochemie entnommen und erleichtert somit dem fachlich versierten Benutzer die Bedienung. Diese Metapher hat aber auch Vorteile für die Umsetzung in computergraphische Technologie. So sind durch die fixe Montage der Handschuhe die Freiräume für die Bewegung von Händen und Armen hinreichend eingeschränkt und für den Benutzer einer Glove Box auch üblich. Dies erleichtert das Registrieren und Verfolgen der Bewegungen des Benutzers.

Die Integration des Visualisierungs- und Interaktionskonzepts in das Konzept des Kopplungsagenten ist aufgrund der Vorüberlegungen nahtlos.

Das vorgestellte Gesamtkonzept wurde in einer Reihe von konkreten Realisierungen umgesetzt, die insbesondere die neuen Möglichkeiten der räumlichen Simulation und Visualisierung validieren.

Neben einer gerätetechnischen Hardwarerealisierung der *Virtual Glove Box* lag der Schwerpunkt der Realisierungen bei den entwickelten Softwareapplikationen.

Es wurde eine graphische Benutzungsschnittstelle implementiert, die die Möglichkeiten einer haptischen Darstellung mit der visuellen Darstellung verbindet. Aus dem Bereich der computergestützten Biochemie wurden Anwendungen zum Molecular Modeling, zur Systembiologie und zur Simulation und Visualisierung der Zelldifferenzierung entwickelt.

Das Prinzip der Molecular-Modeling-Anwendung modelliert in der höchsten Detailstufe jedes Atom eines Moleküls als eigenständige Entität, die entsprechend durch einen eigenen Agenten gekapselt wird. In geringeren Detailstufen werden funktionale Gruppen, strukturelle Gruppen etc. zusammengefasst und so eine Hierarchie von Komponenten aufgebaut. Die eingesetzten Agenten verbinden gemäß dem vorgestellten Konzept der Kopplungsagenten Simulation, Interaktion und Visualisierung, denn der Benutzer interagiert direkt mit den Simulationskomponenten. Mit dieser Herangehensweise verschmelzen die Vorgänge der Modellierung, der Simulation und der anschließenden Ergebnisvisualisierung zu einem kontinuierlichen parallelen Prozess.

Die drei Anwendungen zur Systembiologie – Zelleditor, Simulationskonfigurator und ein gekoppeltes 3-D-Reaktions-Diffusionssystem als konkrete Simulationsanwendung – zeigen am deutlichsten den Vorteil des in dieser Arbeit vorgestellten Gesamtkonzeptes. Der Zelleditor erlaubt die visuelle Modellierung einer dreidimensionalen virtuellen Zelle, wobei die Simulation zunächst im Hintergrund steht. Die „Bausteine“ der Zelle werden dabei ebenfalls als konfigurierbare Bausteine behandelt. Das Simulationskonfigurationswerkzeug verbindet nun Visualisierung und Simulation durch Elemente der Benutzungsschnittstelle, wie etwa „Spraydosen“ oder „Spritzen“, die ein gezieltes Einbringen virtueller Reagenzien ermöglichen. Ein 3-D-Reaktions-/ Diffusionssystem demonstriert die Möglichkeiten, die das neue räumliche Konzept bietet. In einem Reaktions-/ Diffusionssystem wird im Gegensatz zu einem Reaktionssystem nicht nur die Menge der Reaktionspartner berücksichtig, sondern auch deren räumliche Verteilung und die Veränderung dieser Verteilung durch Diffusion.

Das System zur Simulation und Visualisierung der Zelldifferenzierung zeigt ein Visualisierungssystem, das erstmals 3-D-Simulation direkt mit der dreidimensionalen Visualisierung von Zelldifferenzierungsdaten verbindet. Auch in diesem Anwendungsgebiet fügen sich gekoppelte Komponenten zu einer gemeinsamen Visualisierungs- und Simulationsumgebung. Die Besonderheit des Anwendungsfalls Zelldifferenzierung ist, dass sich, im Gegensatz etwa zur Systembiologie, die Computersimulation als Mittel des

Erkenntnisgewinns noch in der Anfangsphase befindet. So kann in diesem Anwendungsfeld die Entwicklung der 3-D-Visualisierung mit der Entwicklung dreidimensionaler Simulationsmodelle einhergehen, statt der Modellierung nachzufolgen.

6.2. Evaluierung

In diesem Kapitel sollen die vorgestellten Realisierungen mit den Anforderungen aus dem entsprechenden Kapitel verglichen werden. Entsprechend der Reihenfolge, in der die Realisierungen vorgestellt wurden, wird auch hier zunächst die Hardwarerealisierung, die *Virtual Glove Box* und das damit verbundene User-Interface, kritisch untersucht werden, bevor auf die drei Anwendungsbereiche der computergestützten Biochemie eingegangen wird. Den Abschluss bilden die Realisierungen jenseits der computergestützten Biochemie.

Interaktion

Die Hauptanforderung an die Interaktion, wie sie im entsprechenden Kapitel formuliert wurde, ist die Forderung nach einem intuitiven, zielgruppengerechten Zugang zu den verschiedenen Entitäten der Simulation. Die Zielgruppe für die in dieser Arbeit vorgestellten Werkzeuge sind Forscher aus dem Umfeld der computergestützten Biochemie, und dabei vorwiegend Biologen oder Biochemiker. Die *Virtual Glove Box* benutzt eine Metapher aus dem biologisch-chemischen Anwendungsbereich und erlaubt daher auf konzeptioneller Ebene den geforderten intuitiven Zugang.

Bei der Umsetzung des Konzeptes im Zuge der Entwicklung der graphischen Benutzungsschnittstelle mit haptischem Feedback zeigten sich jedoch schwerwiegende Softwarefehler in der Treibersoftware der Cybergrasp-Exoskelette, die die praktische Einsetzbarkeit des Systems einschränken. Leider ließen sich komplexe Objekte nur fehlerhaft „begreifen“. Komplex sind dabei alle Objekte, die nicht aus primitiven Objekten wie Kugeln oder Würfeln aufgebaut sind. Bei der Benutzung des Projektionssystems ergab sich, dass die Projektionsfläche in der gegenwärtigen Konfiguration mit einer Pixelzahl von 1280 x 1024 Bildpunkten zu nah vor dem Benutzer platziert ist.

Gemäß dem Ansatz der adaptierbaren Interaktionskomponente (siehe noch einmal Abbildung 69) wurde als Ersatzeingabegerät die konventionelle Maus benutzt. Damit wurde der Ansatz der *Entitäten*, Komponenten mit eigenem Simulations- und Visualisierungsverhalten, bestätigt. Aspekte der Interaktion sind integraler Bestandteil aller weiteren Anwendungen und werden dort entsprechend besprochen.

Molecular Modeling

Die Molecular-Modeling-Anwendung setzt das Konzept der *Entitäten* sowohl für Simulation als auch für Visualisierung und Interaktion um. Für kleine bis mittlere Moleküle (< 100 Atome) ist die Anwendung interaktiv benutzbar, darüber hinaus nimmt die Performanz jedoch aufgrund der quadratischen Komplexität des Ansatzes ab (alle Atome interagieren theoretisch mit allen anderen Atomen eines Moleküls). Interaktive Veränderungen sind aber gerade der Eckstein des Konzepts, das hier praktische Grenzen findet.

Die Molecular-Modeling-Anwendung implementiert eine lokale Optimierungsstrategie: Durch die kontinuierliche Simulation ist zu jedem Zeitpunkt ein lokal optimaler Zustand erreicht. Dieses Konzept der lokalen Kontrolle hat gegenüber einer globalen Strategie eine Einschränkung. Lokale energetische Optimierungen werden in der vorgestellten Anwendung also *en passant* gemacht, allein durch die dauernde Simulation der Kräfte, die für die chemischen Verbindungen verantwortlich sind. Derartige Optimierungen führen zu lokalen Minima, die gegebenenfalls weit vom gesuchten globalen Minimum entfernt sind. Durch Verfahren wie das *Simulated Annealing* [177] wird versucht, auf globaler Ebene diese

Nachteile zu mildern und die Chance zu erhöhen, dass das gefundene lokale Minimum dem gesuchten globalen Minimum entspricht. Derartige Lösungsstrategien lassen sich nur schwer mit dem hier vorgestellten Konzept in Einklang bringen und wurden nicht implementiert.

Systembiologie

Der Hauptansatz der gekoppelten Simulatoren ist besonders für dieses Anwendungsgebiet geeignet. Es muss besonders darauf hingewiesen werden, dass die vorgestellten Anwendungen keine Präferenz für ein bestimmtes Simulationsverfahren enthalten. Die Frage nach der Auswahl eines geeigneten Simulationsverfahrens lässt sich nicht abschließend beantworten. Derzeit ist dies vor allem eine Entscheidung zwischen deterministischen oder stochastischen Verfahren; welches konkrete Verfahren verwendet wird, steht dabei im Hintergrund. Beide Verfahren sind nicht für alle Anwendungsfälle gleich gut geeignet. So ist ein deterministisches Verfahren nicht in der Lage, „Ausreißer" zu erzeugen, die aber von besonderer Bedeutung sind, wenn vergleichsweise wenige Entitäten eines Typs im Modell vorhanden sind. Dagegen sind deterministische Ansätze für eine große Zahl von Entitäten angemessener.

Es gibt eine Reihe von Ansätzen, die versuchen, eine Brücke zwischen stochastischer und deterministischer Simulation zu schlagen, so genannte Hybrid-Simulatoren bzw. *Multi-Algorithem-Approaches* [109]. Keiner dieser Ansätze ist bis jetzt endgültig erfolgreich. Anders als die genannten Anwendungen versucht die in dieser Arbeit vorgestellte Anwendung nicht, die verschiedenen Simulationstechniken in *einem* Simulator zu vereinen; vielmehr wird hier die *Kopplung* verschiedener Simulatoren beschrieben. Dies geht einher mit der Adressierung des Modellsubjekts als dreidimensionales Gebilde.

Zelldifferenzierung

Die vorgestellte Anwendung zur Zelldifferenzierung, *D-VISION*, versteht sich als *Vorschlag* an die Entwicklung neuer Simulatoren und Simulationstechniken. Dies ist wichtig, da es in diesem Gebiet bisher wenige Ansätze zur Simulation von Zelldifferenzierungsvorgängen im dreidimensionalen Raum gibt.

Die vorgestellten Datenstrukturen folgen dem in dieser Arbeit entworfenen Grundkonzept. Von bekannten Simulatoren wurde dabei die grundlegende Idee berücksichtigt, Zellen durch enthaltene Substanzen und aktive Gene zu unterscheiden. Prinzipiell wurde bei der Entwicklung davon ausgegangen, dass ein zukünftiger Simulator, der mit *D-VISION* zusammenarbeiten könnte, diese Vorgehensweise als Grundlage nutzt. Das Visualisierungssystem wurde daher darauf ausgelegt, auf die Zustände von Zellen besonderen Wert zu legen.

Die Zellen werden als 3-D-Kugel dargestellt. Um eine realistischere Darstellung der Zellen zu erreichen, wurden die Kugeln deformiert. Neben der Deformation bietet vor allem die eingesetzte Schnittebenen-Technik eine hohe Vergleichbarkeit mit zweidimensionalen Lichtmikroskopiebildern. Dagegen entspricht die dreidimensionale Ansicht eher solchen Bildern, wie sie Elektronenmikroskope liefern.

Neben den Zellen selbst, die in ihrer Form so realistisch wie möglich aussehen sollten, liegt ein besonderes Augenmerk auf der Darstellung von Substanzen und Konzentrationen. Die einzelnen in der Zelle ablaufenden Prozesse werden durch die zeitliche Änderung des Zellzustandes sichtbar. Der Zustand kann über verschiedene Interaktionsmöglichkeiten exploriert werden.

Anwendungen außerhalb der computergestützten Biochemie

Die vorgestellten Anwendungen außerhalb der computergestützten Biochemie (G*läserner Reaktor*, *ETOILE* und Anwendungen aus *SpacemantiX)* wurden, soweit die Projekte abgeschlossen sind (G*läserner Reaktor, ETOILE*), im Projektkontext evaluiert.

Der erstellte *Gläserne Reaktor* wurde im direkten Auftrag der beteiligten Unternehmen und mit diesen zusammen erstellt. Das Anwendungssystem wurde auf mehreren Messen erfolgreich eingesetzt. Gleichzeitig war das Konzept bei der Umsetzung des Reaktors für den Laptop erfolgreich. Die über einen Agenten gekoppelte Schnittstelle zur Simulation erwies sich als sehr robust gegenüber Änderungswünschen der Auftraggeber im Bereich der Visualisierung.

Im Rahmen des *ETOILE*-Projektes wurden von jedem der Anwendungspartner, Metro von Bilbao und Iberdrola, mehrere Trainingsszenarien entwickelt. Im Falle des Trainings für Notfallsituationen in Kernkraftwerken hat eines der erstellten Szenarien eine Dauer von mehreren Stunden. Der Erfolg des Projektes mit dem Ansatz der gekoppelten Agenten für „Mitspieler“ und Umgebungssimulation ist zweifach:

Zum einen ist es gelungen, die Anwender, also die Experten aus den Bereichen Kraftwerk bzw. U-Bahn-Betrieb, in die Rolle der Autoren von Agentenverhalten zu bringen und so direkt Trainingsszenarien zu gestalten und durchzuführen. Zum anderen ließ sich mit Hilfe des ETOILE-Systems das Training für Notfallsituation tatsächlich verbessern, die Szenarien erlaubten ein entsprechend komplexes Training.

Das Projekt *SpacemantiX* befindet sich zurzeit in der zweiten Hälfte des Projektzeitraums. Damit lässt sich noch keine endgültige Aussage zur Akzeptanz der entwickelten Werkzeuge in der Anwendung machen. Die Identifikation von Entitäten und die Kapselung des Verhaltens und der Visualisierung dieser Entitäten erwiesen sich als hilfreiche Basis für die schnelle und flexible Umsetzung der speziellen SpacemantiX-Werkzeuge.

6.3. Offene Punkte

Es war kaum zu erwarten, dass im Rahmen einer einzigen Dissertation alle Fragestellungen endgültig beantwortet werden konnten. Das gilt insbesondere bei einem so breiten und heterogenen Gebiet wie der Frage der interaktiven Simulation und Visualisierung in der computergestützten Biochemie. Folglich mussten auch in dieser Arbeit einige Punkte offen bleiben. Das besondere Problem einer Arbeit, die das Gebiet der computergestützten Biochemie aus Sicht der Informatik oder genauer aus Sicht der Simulation und schwerpunktmäßig der Computergraphik betrachtet, liegt in der Tatsache, dass sich das Anwendungsfeld der Biochemie gegenwärtig in einem Umbruch befindet und erst jetzt (computer-) graphische Methoden etabliert werden. Im Folgenden sollen einige Punkte aufgegriffen werden, die aus Sicht des Autors dringend einer weiteren Lösung bedürfen.

Die offenen Punkte im Bereich *Interaktion*, speziell der *Virtual Glove Box*, sind vor allem technischer, nicht grundsätzlicher Natur: Die Technik der Exoskelette für Hände und Arme muss verbessert werden. Neben einer Verbesserung der Treibersoftware muss vor allem eine Lösung für das zweihändige Greifen gefunden werden. Beim zweihändigen Greifen werden Gegenstände mit beiden Händen aus gegenüberliegenden Seiten berührt. Die gegenwärtige Technik der Exoskelette ergibt zwar eine Gegenkraft auf die Finger jeder Hand, es gibt aber keinen Mechanismus, eine Kollision der beiden Hände zu verhindern. Eine Gegenkraft für die Arme ist notwendig (siehe Abbildung 127). Eine mögliche Lösung ist eventuell die Verbindung jeder Hand mit einem Roboterarm.

Ein weiterer Punkt ist der erwähnte zu geringe Abstand des Nutzers von der Projektionsfläche. Dieses Problem kann wahrscheinlich ebenso wie die Größe des Prototyps durch die allgemein

zu beobachtende Miniaturisierung der genutzten Komponenten, Projektoren, Displays und haptischen Force-Feedback-Geräte gelöst werden.

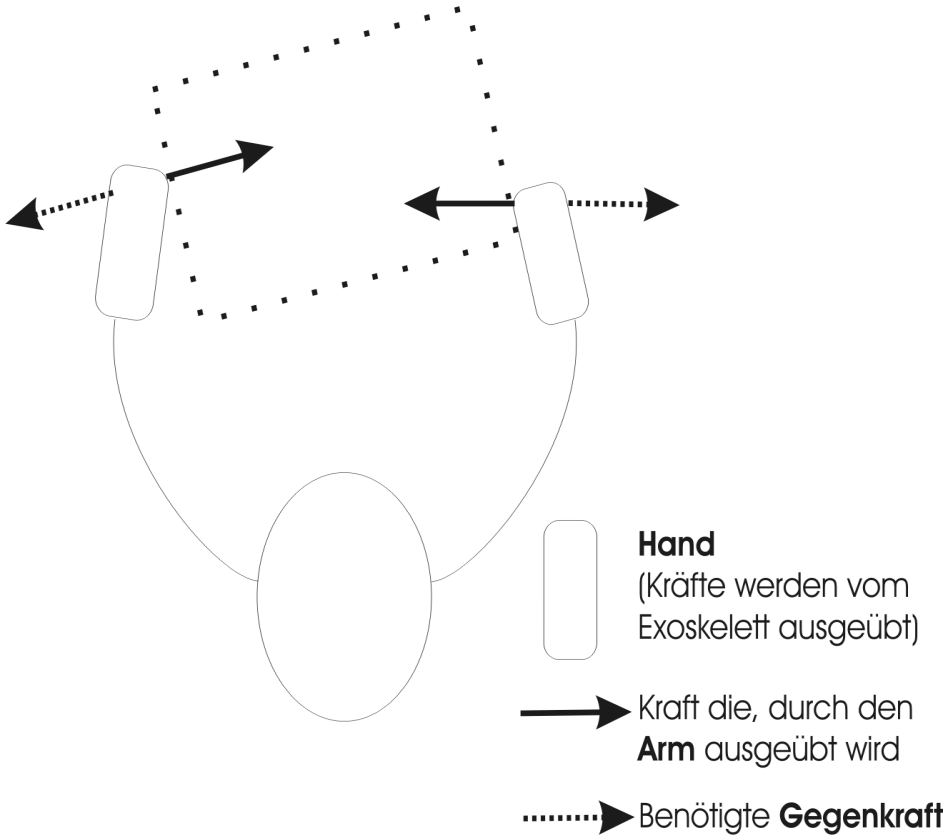


Abbildung 127 Beidhändige Interaktion

Für den Molecular-Modeling-Anwendungsfall bleibt offen, inwieweit sich das neue Paradigma für größere Moleküle eignet. Es fehlt noch ein überzeugender Ansatz, der es erlaubt, zwischen lokaler und globaler Kontrolle „umzuschalten“. Dieses Umschalten lässt sich zwar pragmatisch realisieren, es fehlt aber an einem konzeptionellen Unterbau, der es erlaubt, mit Hilfe agentensprachlicher Mittel das Agentenkonzept auf Zeit zu unterbrechen, die Autonomie der einzelnen Entitäten aufzuheben und sie später wiederherzustellen. Wenn das geschieht, ist es möglich, „alte“ (Analyse- und Simulations-) Module leichter zu integrieren.

Die Systembiologie befindet sich im Moment im Aufbau, sowohl was die Etablierung neuer Simulationstechnologien angeht als auch was die grundsätzliche Akzeptanz dieser Technologie im Kontext der Zellbiologie betrifft. Die Grundannahme dieser Arbeit ist es, dass es benutzungsfreundlicher Werkzeuge bedarf, um die generelle Akzeptanz zu erhöhen, und dass nur spezialisierte Autorenwerkzeuge die hier vorgestellten neuen Paradigmen etablieren können. Die in dieser Arbeit vorgestellten Konzepte und Realisierungen sind ein Anfang in dieser Entwicklung, ihre Grenzen müssen sich in zukünftigen Anwendungen zeigen. Dies gilt insbesondere für den Bereich der Modellierung, denn selbst wenn die Dreidimensionalität des Modellsubjekts für Simulation und Visualisierung handhabbar wird, bedarf es neuer Konzepte, auch die entsprechenden Modelle zu finden bzw. bestehende Modelle zu adaptieren. Dazu gehört auch die Tatsache, dass räumliche Aufteilung des betrachteten Volumens nicht

notwendigerweise reguläre Gitterstrukturen bedeuten muss. Die in dieser Arbeit vorgestellten Konzepte, etwa zur Diffusions-/ Reaktionssimulation, zeigen vielmehr, wie Raumaufteilungen realisiert werden können, die sich an den tatsächlichen räumlichen Strukturen der Modellsubjekte orientieren.

Das eben zur Systembiologie Gesagte hat in noch stärkerem Maße Gültigkeit für den Bereich der Zelldifferenzierung und der Zellverbände. Zwar sind auch hier Simulationsansätze seit Jahren bekannt, die Akzeptanz der Methoden ist jedoch noch gering. Stärker als in den anderen Bereichen fehlt es in diesem Bereich an Simulatoren und Modellen, weil eine passende Visualisierung fehlt. Diese Visualisierung kann aber nur auf die Anforderungen der Simulation reagieren. Die Arbeiten, die in dieser Schrift vorgestellt wurden, sollen diesen „Teufelskreis“ durchbrechen und sowohl für die Visualisierung als auch für die Simulation einen Rahmen bieten. Zu diesem Zeitpunkt muss offen bleiben, ob dieser Ansatz letztendlich erfolgreich ist.

6.4. Zusammenfassung

Die Evaluierung der Konzepte und Realisierungen dieser Arbeit ergibt, dass die Annahme gemeinsamer Strukturen in der computergestützten Biochemie gerechtfertigt war. Das erarbeitete Konzept ließ sich erfolgreich auf alle beschriebenen Bereiche des gesamten Anwendungsgebiets übertragen und entsprechend umsetzen. Durch Anwendungen auch außerhalb des Rahmens der computergestützten Biochemie wurde verdeutlicht, dass sich die Konzepte, zumindest in Teilen, auch auf Simulationsaufgaben außerhalb dieses Bereichs erfolgreich ausdehnen lassen.

Auch im Rahmen dieser Arbeit bleiben Fragen offen, die auf eine Lösung warten. Im Bereich Interaktion sind hierbei vor allem technische Hindernisse zu überwinden. Im Bereich *Molecular Modeling* verlangt ein durch starke kommerzielle Anbieter etabliertes Simulations- und Visualisierungsparadigma nach Lösungen, die eine Migration zu dem in dieser Arbeit vorgestellten neuen Paradigma erleichtern.

Der Bereich der Systembiologie ist im Aufbruch, die hier vorgestellten Arbeiten sind daher eher als Anfang denn als Ende einer Entwicklung zu sehen.

Gleiches gilt für den Bereich der Zelldifferenzierung und der Zellverbände. Die vorliegende Arbeit versucht, ein „Henne-Ei“-Problem zu lösen, indem die erarbeiteten Konzepte zum Rahmen einer weiteren Entwicklung werden können.

Obwohl einige Punkte offen geblieben sind und diese Arbeit nicht für sich in Anspruch nehmen kann, die Probleme der computergestützten Biochemie ein für alle Mal gelöst zu haben, zeigen sich die erarbeiteten Konzepte als robust gegenüber den Anforderungen der verschiedenen Anwendungsfälle.

7. Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurden Probleme der interaktiven Simulation und Visualisierung in der computergestützten Biochemie behandelt, ein Lösungskonzept und dessen Umsetzung vorgestellt. Die besondere Perspektive dieser Arbeit ist die eines *Werkzeuglieferanten,* also eines *Dienstleisters,* der computergraphische Werkzeuge mit simulationstechnischen Konzepten zu einem Rahmen verbindet, der im Kontext des Anwendungsgebietes Biochemie validiert wird. Die Grundannahme dieser Arbeit ist, dass es in den Teilbereichen des Anwendungsgebiets ähnliche Problemstrukturen gibt, deren Gemeinsamkeiten ausgenutzt werden können. Auf diese Weise werden entgegen der gegenwärtig in der Biochemie vorherrschenden Philosophie Synergien nutzbar, die bisher ignoriert werden.

Zu diesem Zweck wurden zunächst in Kapitel 2 die Grundlagen der vorliegenden Arbeit dargestellt. Da die Agententechnologie eine zentrale Rolle im später vorgestellten Konzept spielt, wurde dieser Bereich entsprechend ausführlich dargestellt. Das besondere Augenmerk lag dabei auf der Darstellung des Standes der Technik im Bereich der Agentenkommunikationssprachen. Die Grundlagen zum Thema Simulation sind notwendig zum Verständnis der Probleme, die mit der Simulation als Technik des Erkenntnisgewinns im Allgemeinen verbunden sind. Dies gilt insbesondere für das Verständnis des speziellen Problems der Kopplung verschiedenartiger Simulationstechniken. Die Grundtechniken der Visualisierung sind ebenfalls notwendig für das Verständnis der späteren Resultate dieser Arbeit. Darüber hinaus wurde dem Thema *Interaktion* ein spezielles Kapitel gewidmet, denn das direkte Eingreifen des Nutzers in Simulation und Visualisierung ist ein wichtiger Bestandteil des später vorgestellten Konzeptes. Das Anwendungsgebiet selbst, die computergestützte Biochemie, wird ebenfalls in einem Unterkapitel beleuchtet. Natürlich ist die Darstellung selektiv und betrachtet dieses Gebiet eher aus dem Blickwinkel der Informatik, weniger aus dem der Biochemie.

Im dritten Kapitel wurden die Anforderungen analysiert, die die computergestützte Biochemie stellt. Zu diesem Zweck wurden drei essentielle Teilbereiche der Biochemie hervorgehoben, anhand deren die späteren Konzepte und Realisierungen validiert wurden. Die drei Bereiche, *Molecular Modeling*, *Systembiologie* und *Zelldifferenzierung*, wurden gewählt, da hier bereits im besonderen Maße Simulation als Technik eingesetzt wird – ein Vorgehen, das ansonsten in der Biochemie bisher selten Verwendung findet. Folgerichtig wurden zunächst Anforderungen untersucht, die die computergestützte Biochemie an die Simulation und damit an ein neues Konzept in diesem Bereich stellt. Anschließend wurden ebenso die Anforderungen an den zweiten wichtigen Bereich dieser Arbeit betrachtet, den Bereich der Interaktivität und der Visualisierung. Um bereits zu diesem Zeitpunkt die weitere Anwendbarkeit der Anforderungsbetrachtung und der später vorgestellten Konzepte zu demonstrieren, wurden schließlich Anforderungen betrachtet, die sich aus anderen Anwendungsbereichen ergeben. Zugleich wurde damit ein Hinweis gegeben, auf welche Art die Konzepte in anderen Bereichen Anwendung finden können.

Das Kapitel 4 stellt die in dieser Arbeit entwickelten Kernkonzepte vor. Die Grundannahme der Konzepte besagt, dass es gelingt, verschiedene Aspekte, genannt *Entitäten*, in allen betrachteten Bereichen zu identifizieren, die sowohl für die Gestaltung der (räumlichen) Simulation als auch für die der Interaktivität und der Visualisierung vorteilhaft sind. Im Konzept für die Simulation wird diese Grundannahme als ein System gekoppelter Simulatoren interpretiert. Die Kopplung erfolgt hierbei über ein Multi-Agentensystem. Kernstück dieser kommunizierenden Agenten sind ein auf der FIPA-ACL basierender „Dialekt“, die *BioSLang*, und eine besondere XML-basierte Inhaltsbeschreibungssprache, die *BiSCeD*. Das Konzept zur Interaktion und Visualisierung greift das System der gekoppelten Agenten auf und verwendet es zur

Visualisierung. Das Interaktionskonzept sieht weiterhin ein neues computergraphisches Ein-/ Ausgabegerät vor. Dieses Gerät, die *Virtual Glove Box*, verbindet die visuelle Ausgabe mit einer haptischen Ausgabe auf die Hände des Benutzers, der wiederum mit den Händen direkte Eingaben machen kann. Die Metapher der „Glove Box“ ist direkt dem (erweiterten) Anwendungsgebiet entnommen und erleichtert so den intuitiven Zugang für die Zielgruppe, Forscher in Biologie und Chemie, zu den vorgestellten Methoden. Die beiden Konzepte wurden schließlich zu einem Ganzen integriert.

Im fünften Kapitel wurden die aus den Konzepten resultierenden Implementierungen vorgestellt. Zunächst wurden dabei die Hardware-Realisierungen, d. h. die Arbeiten zur *Virtual Glove Box*, dargestellt, daran anschließend das entsprechende Softwareframework eines graphischen Benutzungsinterfaces mit haptischer Unterstützung. Zu den Realisierungen in Software zählen auch die verschiedenen Anwendungen in den drei betrachteten Teilbereichen der computergestützten Biochemie. Das Kapitel endet mit Anwendungen außerhalb der computergestützten Biochemie, die Konzepte aus dieser Arbeit aufnehmen und im jeweiligen Zusammenhang interpretieren.

Im Kapitel „Evaluierung“, dem sechsten Kapitel, wurden die Realisierungen gegen die vorher beschriebenen Anforderungen bewertet. Hierzu wurden wiederum die verschiedenen Realisierungen in Hard- und Software betrachtet. Bei der Evaluierung wurden einige offene Punkte identifiziert, die auch gleichzeitig einen Ausblick auf zukünftige Arbeiten eröffnen.

Zukünftige Arbeiten müssen die Technik der *Virtual Glove Box* erweitern, um die identifizierten Missstände aufzuarbeiten. Wichtiger ist jedoch die Betrachtung der Auswirkungen auf die Anwendungsdisziplin, die diese Arbeit haben kann. Im Bereich des Molecular Modeling bedarf es besonders einer weiteren Untersuchung, wie bestimmte globale Optimierungsmethoden mit den in dieser Arbeit vorgestellten Konzepten integriert werden können. Ein wichtiges Thema, das von dieser Arbeit kaum berührt wurde, ist die Frage nach einer besonderen computergraphischen Unterstützung des Modellierungsprozesses für dreidimensionale Modelle (gemeint sind biologische Simulationsmodelle, nicht geometrische Modelle).

Das für die Systembiologie Vorgetragene gilt in verstärktem Maß für Zellverbände und die Zelldifferenzierung, da sich dort die Simulation noch nicht im gleichen Grad wie in der Systembiologie etabliert hat.

Wie die Anwendungen jenseits der computergerstützten Biochemie zeigen, lässt sich das Konzept der Kopplungsagenten für heterogene Simulationen auch auf andere Bereiche übertragen. Hier können genauso neue Anwendungsgebiete gesucht werden wie innerhalb der computergestützten Biochemie. Wo Letztere an die Disziplinen der Medizin angrenzt, finden sich logische Erweiterungen auch in dieser Lebenswissenschaft.

8. Literaturverzeichnis

[1] Berg J. M., Tymoczko J. L., Stryer L., Biochemie, Heidelberg, 2003, ISBN 3827413036

[2] http://www.mwa.nrw.de/technologiefuehrer/technologiefelder/life_sciences-inh.htm, besucht am 2004-06-21.

[3] Wiechert W, Modellierung und Simulation metabolischer Netzwerke**,** in: P. Schwarz (Ed.), ASIM-Fachtagung Simulation technischer Systeme, 2001

[4] Dörner, R.; Grimm, P.; Seiler, C, Spezifikation von Verhalten in VR und Computerspielen, In: Bauknecht, Kurt (Hrsg.) u.a.; Gesellschaft für Informatik (GI) u.a.:Informatik 2001. Tagungsband der GI / OCG Jahrestagung : Wirtschaft und Wissenschaft in der Network Economy - Visionen und Wirklichkeit, Wien, 2001, S.1231-1238

[5] Jennings N. R., Faratin P., Lomuscio A. R., Parsons S., Sierra C. and Wooldridge M., Automated negotiation: prospects, methods and chal-lenges, International Journal of Group Decision and Negotiation 10 (2) pg.199-215, 2001

[6] Knapik M. and Johnson J., Developing Intelligent Agents for Distributed Systems, Computing McGraw-Hill, 1998

[7] Parunak H. D., Blue-Collar Agents: Experiences and Issues in the De-velopment and Deployment of Industrial Agent-Based Systems, in the Proceedings of the Fourth International Conference on the Practical Ap-plication of Intelligent Agents and Multi-Agent Technology (PAAM99), pg.3-9, The Practical Application Company LtD., 1999

[8] Fayad M. and Schmidt D.C., Object-oriented application frameworks, in Communications of the ACM, Volume 40, Issue 10 (October 1997), pp. 32-38, 1997, ISSN:00010782

[9] Brooks R. A., Intelligence without reason, Proceedings of IJCAI-91, pg. 569-595, San Mateo, CA: Morgan Kaufmann, 1991

[10] Newell A. and Simon H., Computer science as empirical inquiry: symbols and search, Communications of the ACM, 19(3): 113-126, 1976

[11] Rao et al, 1995]: Rao A. and Georgeff M., BDI Agents: from theory to practice, in Proceed-ings 1st International Conference on Multi-Agent Systems (ICMAS-95), San Francisco, USA, pg. 312-319, 1995

[12] Georgeff M. P. and Lansky A. L., Reactive reasoning and planning. In Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87), pg. 677-682, Seattle, WA, 1987

[13] Burmeister B. and Sundermeyer K., Cooperative problem solving guided by intentions and perception. In Werner E. and Demazeau Y., editors, De-centralized AI 3 – Proceedings of the Third European Workshop on Mod-eling Autonomous Agents and Multi Agent Worlds (MAAMAW-91), pg. 77-92, Elsevier Science Publishers B. V.: Amsterdam, The Netherlands, 1992

[14] Muller J. P., Pischel M., Thiel M., Modelling reactive behaviour in verti-cally layered agent architectures. In Wooldridge M. and Jennings N. R., editors, Intelligent Agents: Theories, Architectures and Languages (LNAI Volume 890), pg. 261-276, Springer-Verlag, Heidelberg, Germany, 1995

[15] Brenner W., Zarnekow R., Wittig H., Intelligente Softwareagenten, Berlin, 1998, ISBN 3540634312

[16] Brazier F. M. T., Dunin-Keplicz B. M., Jennings N. R., and Treur J., DE-SIRE:

Modelling Multi-Agent Systems in Compositional Formal Frame-work, in International Journal of Co-operative Information Systems, 6 (1), 67-94, 1997
[17] Iglesias C. A., Garijo M., Gonzalez J. C., A Survey of Agent-Oriented Methodologies, In Proceedings of the Workshop on Agent Theories, Architectures and Languages, France, 1998
[18] Wooldridge M., Intelligent Agents, In G. Weiss (editor), Multiagent Sys-tems, The MIT Press, April 1999
[19] Maes P. (Hrsg.), Designing Autonomous Agents – Theory and Practice from Biology to Engineering and Back, MIT Press, 1990
[20] Wooldridge M., Agents and software engineering, In AI*IA Notizie XI(3), pages 31-37, September 1998
[21] Zambonelli F., Jennings N. R., and Wooldridge M., Organisational rules as an abstraction for the analysis and design of multi-agent systems, International Journal of Software Engineering and Knowledge Engineering, 2001
[22] Wooldridge M., Jennings N. R., and Kinny D., The Gaia Methodology for Agent-Oriented Analysis and Design, Journal of Autonomous Agents and Multi-Agent Systems 3 (3) 285-312, 2000
[23] Carmel D. and Markovitch S., "Learning Models of Intelligent Agents", Proceedings of AAAI-96, AAAI Press / The MIT Press, pp62-67, 1996
[24] Carver N., Cvetanovic Z. and Lesser V., "Sophisticated Co-operation in FA/C Distributed Problem Solving Systems", Proceedings of 9th National Conference on Artificial Intelligence (AAAI-91), AAAI Press / The MIT Press, pp191-198, 1991
[25] Zhongyan L., WenHuang L. and Lin L., "Agent-based Information Pro-cessing System Architecture", Proceedings of FUSION'99, Sunnyvale, USA, Omnipress, pp. 877-884, 1999
[26] Thomas, S.R., The PLACA Agent Programming Language, in: Woolridge, M, Jennings, N. R. (Hrsg.), Intelligent Agents – Theories, Architectures, and Languages, Lecture Notes in Artificial Intelligence 890, Springer Verlag, Heidelberg, 1995
[27] Smith, R. G., The contract net protocol: High-level communication and control in a distributed problem solver, in IEEE Transactions on Computers, 29 (1980) 12, S.1104-1113
[28] Sycara, K. Decker K., Pannu, A., Williamson, M., Zeng, D., Distributed Intelligent Agents, in IEEE Expert 1996
[29] Hogg L. M., Jennings N. R., Socially Rational Agents, in Proceedings of AAAI Fall Symposium on Socially Intelligent Agents, Mass., Boston, pg. 61-63, 1997
[30] Aumann, R. J. and S. Hart, eds., Handbook of Game Theory with Economic Applications, Volume 1. Amsterdam: North-Holland, 1992
[31] Austin, J. L., How to Do Things With Words, Oxford University Press, Oxford, England, 1962
[32] Searle, J.R., Speech Acts. Cambridge University Press, 1969.
[33] Finin T., Labrou Y. and Mayfield J., KQML as an agent communication language, in Bradshaw J., Software Agents, MIT Press, Cambridge, 1997
[34] http://www.fipa.org, besucht am 2004-06-21
[35] http://logic.stanford.edu/kif/Hypertext/kif-manual.html, besucht am 2004-06-21
[36] Cohen, P.R., Levesque, H. J., Communicative actions for artificial agents, in: Proceedings of the First International Conference on Multi-Agent Systems (ICMAS-95), p. 65-72, San Francisco, CA, USA, Juni 1995
[37] Gruber, T. R., A translation approach to portable ontologies, Knowledge Acquisition, 5(2): 199-220, 1993
[38] Gruber T. R., Toward principles for the design of ontologies used for knowledge

sharing, presented at the Padua workshop on Formal Ontology,
[39] http://www.net-lexikon.de/Rapid-Prototyping.html, besucht am 2004-06-21
[40] Sycara, K., Paolucci, M., van Velsen, M. and Giampapa, J., The RETSINA MAS Infrastructure, 2002, JAAMS.
[41] T. Wagner, A. Garvey, and V. Lesser. Complex goal criteria and its application in design-to-criteria scheduling. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, Providence, July 1997.
[42] Graham, J. R., Decker , K. S., Mersic M., DECAF - A Flexible Multi Agent System Architecture, in Proceedings AAMAS, Amsterdam, 2001
[43] http://www.javasoft.com, besucht am 2004-06-21
[44] http://herzberg.ca.sandia.gov/jess/, besucht am 2004-06-21
[45] A library of FIPA communicative acts and requirements for new communicative acts., http://www.fipa.org/specs/fipa00037/ besucht am 2003-10-10
[46] VDI-Gesellschaft Fördertechnik Materialfluss Logistik (Hrsg.), Simulation von Logistik-, Materialfluß- und Produktionssystemen – Begriffsdefinitionen, Düsseldorf, 1996
[47] Hirsig, R. , Systemtheorie in den Sozialwissenschaften. Skript zur Lehrveranstaltung, 2. überarbeitete Ausgabe. Universität Zürich, 1994
[48] Peter C. Hägele, Physik – Weltbild oder Naturbild?, Tagung der Deutschen Physikalischen Gesellschaft (DPG) Sitzung LTR4 der Lehrertage am 31.03.2000 in Regensburg, 2000
[49] F. Liebl. Simulation - Problemorientierte Einführung. Oldenburg Verlag 1992
[50] Werner Zimmermann: Operations Research, Oldenbourg Verlag, München 1990
[51] Lehn J, Wegmann H, Einführung in die Statistik, Stuttgart, 1992, ISBN 3519120712
[52] L'Ecuyer, Random numbers for simulation, Communications of the ACM, Volume 33 ,Issue 10, pp. 85 – 97, New York, 1990
[53] Wiechert W, Modellkonsistenz und Konfliktlösung bei Hybriden Modellansätzen, in 11. Symposim Simulationstechnik, pp. 643-648, Dortmund, 1997
[54] http://www.mathworks.com/, besucht am 2004-06-21
[55] http://www.aspentech.com/ , besucht am 2004-06-21
[56] Willi Törnig, Peter Spellucci, Numerische Mathematik für Ingenieure und Physiker, Bd.2, Numerische Methoden der Analysis, Springer Verlag , 1990, ISBN 3540518916
[57] Richard S. Wright Jr. und Michael R. Sweet. OpenGL Super Bible, Waite Group Press. 2. Auflage, 1999.
[58] Nilsson N.J., Artificial Intelligence: A New Synthesis, San Francisco, 1998, ISBN 1558605355
[59] Schäfer A, Verhalten von 3D-Ameisen, Diplomarbeit, Frankfurt a. M., 2001
[60] Ben Said L., Bouron T., Drogoul A., , Agent-based interaction analysis of consumer behavior, proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'02), Bologna, Italy (2002)
[61] Davidsson, P.: Agent based social simulation : A computer science view. Journal of Artificial Societies and Social Simulation 5(2002) http://www.soc.surrey.ac.uk/JASSS/5/1/7.html.
[62] A. Drogoul, D. Vanbergue, T. Meurisse , "Multi-Agent Based Simulation: Where are the Agents ?", Proceedings of MABS'02 (Multi-Agent Based Simulation, Bologna, Italy, July 2002), LNCS, Springer-Verlag
[63] Schuhmann H., Müller W., Visualisierung: Grundlagen und allgemeine Methoden, Springer, Berlin Heidelberg, 2000
[64] Prof. Dr. R. Schiedermeier, Computergraphik, 2002. http://www.informatik.fh-muenchen.de/~schieder/graphik-01-02/index.html. besucht am 2004-06-21

[65] Oliver Vornberger und Olaf Müller. Computergrafik, 2000, http://www-lehre.informatik.uni-osnabrueck.de/~cg/2002/skript/node113.html. besucht am 2004-06-21.
[66] Mark de Berg, Marc van Kreveld, Mark Overmars, Otfried Schwarzkopf. Computational Geometry:Algorithms and Applications. Springer, 1997.
[67] Patent US3541541 (USA), "X-Y Position Indicator For A Display System", 1970
[68] A Research Center for Augmenting Human Intellect, Douglas C. Engelbart and William K. English, *AFIPS Conference Proceedings of the 1968 Fall Joint Computer Conference*, San Francisco, CA, 33 (December 1968), pp. 395-410 (AUGMENT,3954,)
[69] Foley, James D., van Dam, Andries, Feiner, Steven K., Hughes, John F., Phillips, Richard L.: Introduction to Computer Graphics. New York: Addison-Wesley, 1997.
[70] Aaron Marcus, Metaphor design for user interfaces, Conference on Human Factors in Computing Systems CHI 98, Los Angeles, California, United States, pp. 129 – 130, 1998
ISBN:1-58113-028-7
[71] Theisinger R., Haptische 3D-Benutzungsoberfläche, Diplomarbeit, Frankfurt a.M., 2002
[72] Anderson, Kognitive Psychologie – Eine Einführung. Spektrum Verlag, 1996
[73] Norman, The Design of Everyday Things, MIT Press, 1988
[74] Gabriele Gramelsberger: Semiotik und Simulation: Die Fortführung der Schrift ins Dynamische (Dissertation, FU Berlin), 2001
[75] http://en.wikipedia.org/wiki/Scene_graph, besucht am 2004-06-21
[76] L. Hitchner. An Introduction to 3D Computer Graphics, (Draft), Kapitel 1, Scene Description and Managing: The Scene Graph.
http://www.csc.calpoly.edu/~hitchner/CSC471.W2003/SceneGraph.pdf, 2000
[77] Weiner, Motivationspsychologie. Beltz Psychologie Verlags Union, 1994
[78] Mundell, Towards a Virtual Operating Environment, http://citeseer.nj.nec.com/mundell99towards.html, 1999
[79] Hinckley, K, Haptic Issues for Virtual Manipulation, Dissertation, University of Virginia, http://citeseer.nj.nec.com/hinckley97haptic.html, 1997
[80] LaViola, Whole-Hand and Speech Input in Virtual Environments, http://citeseer.nj.nec.com/laviola99wholehand.html, 1999
[81] Bowman, Kruijff, LaViola, Poupyrev, An Introduction to 3D User Interface Design. Presence Vol.10, Number 1, 2001, 96-108
[82] Bolt, "put-that-there" - Voice and Gesture at the Graphics Interface. SIGGRAPH '80 Proceesdings, Vol.14, 262-270, July 1980
[83] Balakrishnan, Hickley, Symmetric Bimanual Interaction, CHI 2000, 1-6 April 2000, 33-40
[84] Roessler, Grantz. Performance Evaluation of Input Devices in Virtual Environments, http://vr.iao.fhg.de/papers/3d-input-devices.pdf, kein Jahr
[85] Gribnau, Hennessey, Comparing single- and two-handed 3D input for a 3D object assembly task. CHI'98, Summary, ACM, New York, 1998, pp. 233-234
[86] Zwisler, Rainer, Virtuelle Realität und die Rolle von Haptic, http://www.zwisler.de/scripts/haptics/haptics.html, Juli 1998
[87] Way, Barner. Automatic Visual To Tactile Translation, Part 1: Human Factors, Access Methods and Image Manipulation, IEEE Transactions on Rehabilitation Engineering, March, 1997.
[88] Lederman, Klatzky. Hand movements: a window into haptic object recognition, Cognitive Psychology 19(3), 1987

[89] Newell, Ernst, Tjan, Bülthoff, Viewpoint Dependence in Visual an Haptic Object Recognition. Psychological Science Vol.12, No.1 Januar 2001, 37-42
[90] Hajian, Howe. Biomechanics of Manipulation: Grasping the Task at Hand, in J. Winters and P. Crago, eds, Neural Control of Posture and Movement, Springer-Verlag, 2000, pp.382-389.
[91] Pawluk, Howe. Dynamic Contact of the Human Fingerpad Against a Flat Surface, Journal of Biomechanical Engineering, Volume 121, pp. 605-611, 1999
[92] http://www.websters-online-dictionary.org/, besucht am 2004-06-21
[93] http://www.nih.gov/, besucht am 2004-06-21
[94] http://www.bisti.nih.gov/CompuBioDef.pdf, besucht am 2004-06-21
[95] F. Cramer: Molekulares Erkennen: 100 Jahre Schlüssel-Schloß-Hypothese durch Emil Fischer. Plenarversammlungen. Braunschw. Wiss. Ges. Jahrbuch 1995, Göttingen, 17-18 (1996).
[96] http://www.accelrys.com/cerius2/, besucht am 2004-06-21
[97] http://www.pc.chemie.tu-darmstadt.de/research/molcad/, besucht am 2004-06-21
[98] http://www.web3d.org/x3d/specifications/vrml/ISO_IEC_14772-All/index.html, besucht am 2004-06-21
[99] http://www.rcsb.org/pdb/, besucht am 2004-06-21
[100] http://www.cis.rit.edu/htbooks/nmr/, besucht am 2004-06-21
[101] http://ndbserver.rutgers.edu/mmcif/, besucht am 2004-06-21
[102] Bereiter-Hahn, J.: Akustische Mikroskopie. In: Mikroskopie in Forschung und Praxis, Horst Robenek (Hrsg.). Darmstadt: GIT Verlag, 1995. S. 261_323
[103] http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml, besucht am 2004-06-21
[104] http://www.cellsimulation.de/, besucht am 2004-06-21
[105] B.E. Oleson, M. Möller, K. Prank, 4DiCeS: Four-Dimensional Cell Simulation and Visualization, European Conference on Computational Biology (ECCB 2003), MSB14
[106] Schaff, J., Slepchenko, B., and Loew, L. M., Physiological modeling with the Virtual Cell framework., In Johnson, M. and Brand, L., (Hrsg.), Methods in Enzymology, volume 321, pages 1–23. Academic Press, San Diego. 2000
[107] http://www.nrcam.uchc.edu/, besucht am 2004-06-21
[108] Boris M. Slepchenko, James C. Schaff, John H. Carson, Leslie M. Loew, COMPUTATIONAL CELL BIOLOGY: Spatiotemporal Simulation of Cellular Events, Annual Review of Biophysics and Biomolecular Structure, June 2002, Vol. 31: 423-441
[109] Tomita, M., Hashimoto, K., Takahashi, K., Shimizu, T., Matsuzaki, Y., Miyoshi, F., Saito, K., Tanida, S.,Yugi, K., Venter, J. C., and Hutchison, C., E-Cell: Software environment for whole cell simulation., Bioinformatics, 15(1):72–84, 1999
[110] A Simulation Model of Diabetes Using E-CELL System,Yasuhiro Naito, Hiroshi Ohno, Sae Seno, Yuki Oguchi, Atsumi Sano, Hiromu Nakajima, Masaru Tomita, Genome Informatics 12:310-311, 2001
[111] Towards Simulation of Whole Metabolic Pathways in Human Erythrocyte, Ayako Kinoshita, Yoichi Nakayama, Masaru Tomita, SauGenome Informatics 12:312-313, 2001
[111a] An Electrophysiological Simulation Model of the Myocardial Cell Using E-CELL System, Motohiro Yoneda, Yasuhiro Naito, Shoko Miyamoto, Sayaka Ishinabe, Shinobu Kuratomi, Nobuaki Sarai, Satoshi Matsuoka, Akinori Noma, Masaru Tomita, Genome Informatics 12:314-315,2001
[112] http://www.projectcybercell.com, besucht am 2004-06-21

[113] M. Hucka, A. Finney, Herbert M. Sauro, H. Bolouri, J. Doyle, and H. Kitano. In the Proceedings of the Pacific Symposium on Biocomputing 2002, Jan. 2002.
[114] http://java.sun.com/products/jdk/rmi/reference/whitepapers/javarmi.html, besucht am 2004-06-21
[115] http://www.omg.org/gettingstarted/corbafaq.htm, besucht am 2004-06-21
[116] http://en.wikipedia.org/wiki/RPC, besucht am 2004-06-21
[117] Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, H. Kitano, A. P. Arkin, B. J. Bornstein, D. Bray, A. Cornish-Bowden, A. A. Cuellar, S. Dronov, E. D. Gilles, M. Ginkel, V. Gor, I. I. Goryanin, W. J. Hedley, T. C. Hodgman, J.-H. Hofmeyr, P. J. Hunter, N. S. Juty, J. L. Kasberger, A. Kremling, U. Kummer, N. Le Novere, L. M. Loew, D. Lucio, P. Mendes, E. Minch, E. D. Mjolsness, Y. Nakayama, M. R. Nelson, P. F. Nielsen, T. Sakurada, J. C. Schaff, B. E. Shapiro, T. S. Shimizu, H. D. Spence, J. Stelling, K. Takahashi, M. Tomita, J. Wagner, J. Wang. The Systems Biology Markup Language (SBML): A Medium for Representation and Exchange of Biochemical Network Models, Bioinformatics 19(4):524-531, 2003.
[118] http://www.w3.org/XML/, besucht am 2004-06-21
[119] Morton-Firth, C. J. and Bray, D.. Predicting temporal fluctuations in an intracellular signallingpathway. Journal of Theoretical Biology, 192:117–128, 1998
[120] Kirkwood, T. B. L., Boys, R. J., Gillespie, C. S., Proctor, C. J., Shanley, D. P., and Wilkinson, D. J..Towards an e-biology of ageing: integrating theory and data. Nature Reviews Molecular Cell Biology, 4:243–249., 2003
[121] Belta, C., Goulian, M., Gowen, S., Ivancic, F., Kumar, V., Owen, A., Rubin, H., Rubin, M., Schug, J.,Sokolsky, O., Webb, J., and Welber, L.,, http://bio.bbn.com/biospice/biosketchpad/, 2003
[122] McCollum, M. and Lancaster, J., Biospreadsheet, 2003
[123] Arkin, A. P., Simulac and Deduce, http://gobi.lbl.gov/~aparkin/Stuff/Software.html, 2001
[124] Funahashi, A. and Kitano, H.. CellDesigner, http://www.systems-biology.org/, 2003
[125] Shapiro, B. E., Levchenko, A., and Mjolsness, E., Automatic model generation for signal transduction with applications to MAP-kinase pathways. In Kitano, H., editor, Foundations of Systems Biology, Kapitel 7, pp 145–161. MIT Press, 2001
[126] Mendes, P. New research software to simulate biochemical processes, http://www.vbi.vt.edu/pr/press_releases/press_20001218_news_new-software.htm, 2000
[127] Goryanin, I., Hodgman, T. C., and Selkov, E., Mathematical simulation and analysis of cellular metabolism and regulation. Bioinformatics, 15(9):749–758, 1999
[128] Peterson, G. D. and Drager, S. L., Accelerating defense applications using high performance reconfigurable computing. In Proceedings of 2003 Government Microcircuit Applications Conference (GOMAC), pp. 544–547, Tampa, Florida, USA., 2003
[129] Sauro, H. M. and Fell, D. A., SCAMP: A metabolic simulator and control analysis program. Mathl. Comput. Modelling, 15:15–28, 1991
[130] Sauro, H. S., JDesigner: A simple biochemical network designer, http://members.tripod.co.uk/sauro/biotech.htm, 2001
[131] Vass, M., Shaffer, C. A., Tyson, J. J., Ramakrishnan, N., and Watson, L. T., The jigcell model builder: A tool for modeling intra-cellular regulatory networks. Bioinformatics. 2003
[132] Bartol, T. M. and Stiles, Mcell, http://www.mcell.cnl.salk.edu/, 2002
[133] Schilstra, M. and Bolouri, H., Netbuilder,

http://strc.herts.ac.uk/bio/maria/NetBuilder/index.html, 2002
[134] Minch, E., de Rinaldis, M., and Weiss, S., pathSCOUT: exploration and analysis of biochemical pathways. Bioinformatics, 19(3):431–432, 2003
[135] Stelling, J., Ginkel, M., Bettenbrok, K., and Gilles, E. D, Towards a virtual biological laboratory. In Kitano, H., editor, Foundations of Systems Biology, chapter 5, pages 189–212. MIT Press, 2001
[136] http://www.cellml.org/ besucht am 2004-06-21
[137] http://www.w3.org/Math/, besucht am 2004-06-21
[138] http://www.physiome.com/, besucht am 2004-06-21
[139] http://en.wikipedia.org/wiki/SIMD, besucht am 2004-06-21
[140] Klaus Seidl: Approaching Virtual Organism by PheGe. ECAL 2003: 696-705,2003
[141] Furusawa C., Kaneko K., Emergence of Multicellular Organisms with Dynamic Differentiation and Spatial Pattern, Artificial Life 4: 79–93 , 1998
[142] http://www.cellsimulation.de, besucht am 2004-06-21
[143] Cremer T., Cremer M., Wagler B., Walter J., Schermelleh L., Imaging of human cell nuclei in space and time, Zellbiologie *aktuell,* 29. Jahrgang, Ausgabe 3/2003, Heidelberg
[144] Gillespie, D. T., Exact stochastic simulation of coupled chemical reactions, Journal of Physical Chemistry, 81, 2340-2361, 1977
[145] Gibson, M.A., Computational Methods for Stochastic Biological Systens, Thesis, California Insitute of Technology, Pasadena, USA, 2000
[146] Vasudeva, K. and Bhalla, U. S., Adaptive stochastic-deterministic chemical kinetic simulations, Bioinformatics, Vol. 20 no. 1 2004, pp. 78–84, 2004
[147] Kiehl, T. R., Mattheyses, R. M. and Simmons, M. K., Hybrid simulation of cellular behavior, Bioinformatics, Vol. 20 no. 3 2004, pages 316–322, 2004
[148] http://www.gdv.informatik.uni-frankfurt.de/cellsimulation/speaker.php?speaker=westerhoff, besucht am 2004-06-21
[149] Journal "Apoptosis", http://www.kluweronline.com/issn/1360-8185, besucht am 2004-06-21
[150] Card, S. K., Mackinlay J. D., Shneiderman B., Mckinley J. D., Readings in Information Visualization, San Francisco, 1999, ISBN 1558605339
[151] http://www.chemie.fu-berlin.de/fb/fachdid/kunststoffe/polysons.htm, besucht am 2004-06-21
[152] Seiler, C., Grimm, P., Dörner, R.,: Die Virtual Glove Box: Interaktion mit virtuellen Objekten in Sichtsystemen, In: Möller, Reinhard: 7. Workshop Sichtsysteme - Visualisierung in der Simulationstechnik., Aachen : Shaker, 2001, S. 55-62
[153] Wilson, L.F.; Burroughs, D.J.; Kumar, A.; Sucharitaves, J.;A framework for linking distributed simulations using software agents Proceedings of the IEEE,Volume: 89 ,Issue: 2 ,Feb 2001, Pages:186 – 200
[154] http://www.fipa.org/specs/fipa00079/XC00079B.pdf, besucht am 2004-06-21
[155] Decker, K. S., X. Zheng, C. Schmidt, A Multi-Agent System for Automated Genetic Annotation, in: *Proceedings of the 5th Intl. Conf. on Autonomous Agents*. Montreal, pp. 433–440, 2001
[156] Persönliche Konversation mit Prof. Olaf Wolkenhauer, anläßlich der ACHEMA 2003 [180]und der Vortragsreihe (siehe [106])
[157] http://en.wikipedia.org/wiki/Diffusion, besucht am 2004-06-21
[158] Köykkä, M., R.Ollikainen and M. Ranta-aho, et al., Usability Heuristic Guidelines for 3D Multiuser Worlds. Proceedings of the 1999 Conference on Computer Human Interaction Special Interest Group of the Ergonomics Society of Australia - Wagga Wagga, Australia, November 28-30, 1999. J. Scott and B. Dalgarno, Charles Sturt

University: 52-57,
[159] http://www.robotic.dlr.de/mmi/sm/, besucht am 2004-06-21
[160] Cohen, A. and Chen, E. "Six Degree-of-Freedom Haptic System for Desktop Virtual Prototyping Applications". In Proceedings of the ASME Winter Annual Meeting, Dynamics Systems and Control, DSC-Vol67, pp. 401-402, Nashville, Tennessee, November 1999.
[161] CyberGrasp von Virtual Technologies, Inc: http://www.virtex.com/products/hw_products/cybergrasp.html, besucht am 2000-06-30
[162] http://techpubs.sgi.com/library/tpl/cgi-bin/browse.cgi?coll=nt&db=bks&cmd=toc&pth=/SGI_Developer/Cos3C_PG, besucht am 2004-06-21
[163] Dörner, Ralf, Erstellung und Präsentation von Animationen für Trainingszwecke: das Konzept der Animationsagenten und seine Umsetzung, Dissertation, Johann Wolfgang Goethe-Universität, Frankfurt am Main, 2001
[164] Burns, D. Open Scene Graph, IMAGE 2003, Chandler, AZ, USA, 2003
[165] Gamma, E., Helm R., Johnson, R., Vlissides, J., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1994
[166] http://www.fox-toolkit.org/, besucht am 2004-06-21
[167] http://www4.discreet.com/3dsmax/, besucht am 2004-06-21
[168] Izquierdo Rojas, P., Konzept und Implementierung eines Systems zur Visualisierung von Zelldifferenzierungssimulationen, Diplomarbeit, Johann Wolfgang Goethe-Universität, Frankfurt am Main, 2004
[169] C. Healey: Preattentive Processing, (online), http://www.csc.ncsu.edu/faculty/healey/PP/, 1999
[170] B. Alberts, D. Bray, J. Lewis, M. Raff, K. Roberts, J. D. Watson, Molekularbiologie der Zelle, Dritte Auflage, VCH, Weinheim, 1995
[171] G. M. Nielson, B. Hamann: Techniques for the interactive visualization of volumetric data. IEEE: Proceedings of the 1st conference on Visualization '90, Session: volume visualization techniques, S. 45-50, San Francisco, 1990
[172] A. Stork: Effiziente 3D-Interaktions- und Visualisierungstechniken für benutzer-zentrierte Modellierungssysteme, Fachbereich Informatik der Technischen Universität Darmstadt, Darmstadt, 2000
[173] www.basell.com, besucht am 2004-06-21
[174] Arno Schäfer, „Animations- und Verhaltensbeschrebung in VRMLSzenen“, Diplomarbeit, Technische Hochschule Darmstadt, 1996
[175] http://www.quantum3d.com/products/opengvs/about.htm, besucht 2004-06-21
[176] www.blazesoft.com, besucht am 2004-06-21
[177] S. Kirkpatrick and C. D. Gelatt and M. P. Vecchi, Optimization by Simulated Annealing, Science, Vol 220, Number 4598, pages 671-680, 1983
[178] Seminar Visualisierung und grafische Standards. http://gdv-serv.prakinf.tu-ilmenau.de/SeminarVis4/, besucht am 2004-06-21
[179] http://www.umass.edu/microbio/rasmol/, besucht am 2004-06-21
[180] www.achema.de, besucht am 2004-06-21
[182] Dr. phil. Valentina Hinz und Dipl.-Ing. Stefan Franz. 3D - Computergrafikenfür Archäologie und Bauforschung. http://www.hinzundfranz.de/dt/dtmet.htm, besucht am 30.11.2003.
[183] Woernle, C., Treder, M., Vorlesung zur Simulationstechnik in der Mechatronik, Universität Rostock, 2004
[184] http://www.dev-biologie.de/grundlagen/stammzellen/stammzellen.htm, besucht am 2004-06-21

[185] http://bioinformatics.icmb.utexas.edu/lgl/Images/rsomZoom.jpg, besucht am 2004-06-21
[186] Cal3D, http://cal3d.sourceforge.net/, besucht am 2004-06-21
[187] Shannon C, Communication in the Presence of Noise, Proceedings of the I.R.E, New York, 1949
[188] Duden, Rechtschreibung der deutschen Sprache, Band 1. Dudenverlag Mannheim/Leipzig/Wien/Zürich, 21. Auflage, 1996
[189] http://www.mdl.com/solutions/white_papers/ctfile_formats.jsp, besucht am 2004-06-21
[190] http://java.sun.com/products/java-media/3D/
[191] http://www.sgi.com/products/software/performer/
[192] http://www.opensg.org/
[193] Schmuker, Kurt J. "The Complete Book of Lisa". Harper & Row 1984

8.1. Eigene Veröffentlichungen

8.1.1 Wissenschaftliche Veröffentlichungen

Breiner, Tobias; Dörner, Ralf ; Seiler, Christian ; Gudo, Michael: Visualizing Organisms with Hydraulic Body Parts: A Case Study in Integrating Simulation and Vizualisation Models - Akzeptiert zur Veröffentlichung - Mai 2004 . In: Symposium on Visualisation (2004), S.1-8

S. Hartmann, Christian Seiler, Ralf Dörner, Paul Grimm: Case Study: Visualization of Mechanical Properties and Deformations of Living Cells. in: INFORMATIK 2003 - Innovative Informatikanwendungen, Band 1, Beiträge der 33. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 29. September - 2. Oktober 2003, pp. 366-371, Frankfurt am Main. LNI 34 GI 2003, ISBN 3-88579-363-6

Ralf Dörner, Christian Seiler, Jens Barthelmes, Detlef Krömker: Visualisierung als Gegenstand der Ausbildung von Bioinformatikern. in: INFORMATIK 2003 - Innovative Informatikanwendungen, Band 1, Beiträge der 33. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 29. September - 2. Oktober 2003, pp. 381-383, Frankfurt am Main. LNI 34 GI 2003, ISBN 3-88579-363-6

Seiler, Christian: Component-Based Approach for Molecular Modelling Applications. In: Geiger, Christian (Ed.) u.a.: Structured Design of Virtual Environments and 3D-Components. Proceedings : Workshop at the Web3D 2001 Conference. Aachen : Shaker, 2002, 12 p.

Dörner, Ralf; Grimm, Paul; Seiler, Christian: Spezifikation von Verhalten in VR und Computerspielen. In: Bauknecht, Kurt (Hrsg.) u.a.; Gesellschaft für Informatik (GI) u.a.: Informatik 2001. Tagungsband der GI / OCG Jahrestagung : Wirtschaft und Wissenschaft in der Network Economy - Visionen und Wirklichkeit. Wien : österreichische Computer Gesellschaft, 2001, S.1231-1238

Seiler, Christian; Grimm, Paul; Dörner, Ralf: Die Virtual Glove Box: Interaktion mit virtuellen Objekten in Sichtsystemen. In: Möller, Reinhard: 7. Workshop Sichtsysteme - Visualisierung in der Simulationstechnik., Aachen : Shaker, 2001, S. 55-62

Seiler, Christian; Dörner, Ralf: A Virtual Glovebox for the Digital Enterprise. In: Kovacs, George (Ed.) u.a.; IFIP TC5/WG 5.2 u.a.: Digital Enterprise Challenges : Live-Cycle Approach to Management and Production. Boston; Dordrecht; London : Kluwer Academic Publishers, 2001, pp. 274-283

Dörner, Ralf; Grimm, Paul; Seiler, Christian: Agentenbasierte Simulation menschlichen Verhaltens in Notfallsituationen für Trainingszwecke. In: Schulze, Thomas (Hrsg.) u.a.; Otto-

von-Guericke-Universität Magdeburg: Simulation und Visualisierung. Proceedings 2000. Ghent : SCS-Europe, 2000, S. 211-224

Dörner, Ralf; Grimm, Paul; Seiler, Christian: Agents and Virtual Environments for Communication and Decision Training for Emergencies. In: Sierra, Carles (Ed.) u.a.: Proceedings of the Fourth International Conference on Autonomous Agents. New York : ACM Press, 2000, pp. 50-51

Dörner, Ralf; Grimm, Paul; Seiler, Christian: Ein komponentenbasiertes Basissystem für Digital Storytelling. In: Spierling, Ulrike (Hrsg.): Digital Storytelling - Tagungsband. Stuttgart : Fraunhofer IRB Verlag, 2000, S. 137-147

Dörner, Ralf; Grimm, Paul; Seiler, Christian; Prüße, Michael; Stührmann, Kirsten: Agentenbasierte Simulation mit Sichtsystemen. In: Möller, Reinhard: 6. Workshop Sichtsysteme - Visualisierung in der Simulationstechnik. Aachen : Shaker, 1999, S. 33-44

Dörner, Ralf; Grimm, Paul; Seiler, Christian: Agenten und Mentale Modelle : Neue Wege im Management-und Kommunikationstraining für Notfallsituationen. In: GIT Sicherheit + Management 8 (1999), 6, S. 541-543

Schäfer, Arno; Seiler, Christian: An Author-friendly Concept for Multi-User Virtual Environments in VRML. In: Maurer, Hermann u.a.; Association for the Advancement of Computing in Education (AACE): WebNet 98 - World Conference of the WWW, Internet & Intranet. Proceedings. Bd. 2. Charlottesville, VA, USA : Association for the Advancement of Computing in Education, AACE, 1998, pp. 1177-1178

Seiler, Christian; Schäfer, Arno: MUSyC: Scalablable Multi-User Virtual Environments based on VRML. In: Göbel, Martin u.a.; European Association for Computer Graphics (Eurographics) u.a.: Virtual Environments '98. Proceedings. Wien : Springer, 1998, pp. 26/1-26/9

Alexa, Marc; Gerfelder, Norbert; Grimm, P.; Seiler, Christian: AVWoD - Concept and Realization for Internet-Based Media Integration. In: Hoschka, Philipp; Institut National de Recherche en Informatique et en Automatique (INRIA) u.a.: Real Time Multimedia and the World Wide Web. W3C Workshop. Sophia-Antipolis : Institut National de Recherche en Informatique, 1996, pp. 14/1-14/7

8.1.2 Sonstige Veröffentlichungen

Dörner, Ralf; Grimm, Paul; Seiler, Christian; Westphal, Joachim; Stührmann, Kirsten; Prüße, Michael; Fraunhofer Anwendungszentrum Computergraphik in Chemie und Pharmazie (AGC); STN Atlas Elektronik: The Designs of the Toolset : EU-Projekt - ESPRIT 29036: Deliverable. 2001

Dörner, Ralf; Grimm, Paul; Seiler, Christian; Westphal, Joachim; Stührmann, Kirsten; Fraunhofer Anwendungszentrum Computergraphik in Chemie und Pharmazie (AGC); STN Atlas Elektronik: User's Manual for the Tools : EU-Projekt - ESPRIT 29036: Deliverable. Frankfurt am Main, 2001

Elcacho, Colette; Seiler, Christian; Krömker, Detlef (Abteilungsleiter); Fraunhofer-Institut für Graphische Datenverarbeitung (IGD): Design Report Specifying the Collaborative Functionality of the 3D System : Esprit Project 22005. Darmstadt, 1998

Seiler, Christian; Luckas, Volker (Betreuer); Schäfer, Arno (Betreuer): Mehrbenutzerunterstuetzung fuer VRML 2.0. Darmstadt, 1997, Techn. Univ. Darmstadt, Diplomarbeit, 1997

Seiler, Christian; Gerfelder, Norbert (Betreuer): Codierung und Uebertragung von Audio-Informationen. Darmstadt, 1996, Techn. Hochsch. Darmstadt, Studienarbeit, 1996

Ackermann, Michael; Alexa, Marc; Broll, Tanja; Gerfelder, Norbert; Grimm, P.; Hecker, Cornelius; Höfling, Stefanie; Krömker, Detlef; Müller, Wolfgang (GRIS); Schmücker, Joerg; Seiler, Christian; Spierling, Ulrike; Vohsbeck-Petermann, Ralf; Wiener, Andreas; Zhou, Jian: High Definition Multimedia-Systeme im Office 2000. Abschlussbericht : Zusammenfassung der Arbeiten und Ergebnisse. Darmstadt, 1996

8.1.3 Betreute Arbeiten

Izquierdo Rojas, Paul, Seiler, Christian (Betreuer), Barthelmes, Jens (Betreuer): Konzept und Implementierung eines Systems zur Visualisierung von Zelldifferenzierungssimulationen, Frankfurt 2004, Johann Wolfgang Goethe-Universität, Frankfurt am Main., Dipl.-Arbeit, 2004

Schäfer, Jan, Barthelmes, Jens (Betreuer), Seiler, Christian (Betreuer): 3D computergraphisches

Modellierungswerkzeug für biologische Zellen, Frankfurt, 2003, Fachhochschule Gießen-Friedberg, 2003

Theisinger, Rolf; Seiler, Christian (Betreuer); Baier, Wolfgang (Betreuer): Haptische 3D-Benutzungsoberfläche. Frankfurt, 2002, Johann Wolfgang Goethe-Universität, Frankfurt am Main., Dipl.-Arbeit, 2002

Hartmann, Steffen; Seiler, Christian (Betreuer); Dörner, Ralf (Betreuer); Grimm, Paul (Betreuer): Konzeption und Evaluierung einer Visualisierungstechnik für mechanische Eigenschaften von biologischen Zellen. Frankfurt am Main, 2002, Johann Wolfgang Goethe-Universität, Frankfurt am Main., Dipl.-Arbeit, 2002

Huang, Lidong; Seiler, Christian (Betreuer); Breiner, Tobias (Betreuer): Programme und Skript-Sprache für die Modellierung von Molekülen und die Behandlung ihrer Konformationen. Frankfurt am Main, 2002, Johann Wolfgang Goethe-Universität, Frankfurt am Main, Diplomarbeit, 2002

Walburg, Jan; Grimm, Paul (Betreuer); Seiler, Christian (Betreuer): Visualisierung in der Bioinformatik am Beispiel von Microarrays. Frankfurt am Main, 2002, Heidelberg, Fachhochsch., Dipl.-Arb., 2002

Zerbst, Boris; Grimm, Paul (Betreuer); Seiler, Christian (Betreuer): Konzeption und Implementierung eines 3D-Molekülviewers. Darmstadt, 1999 Darmstadt, Fachhochsch., Dipl.-Arbeit, 1999

9. Anhang

9.1. Beispiel „Confirm“

Anhand des „Confirm“ performatives soll gezeigt werden, wie performatives in [34] definiert wurden.

Formale Beschreibung: *<i, confirm (j,p)>*

i ist der Sender, *j* der Empfänger, *p* die bestätigte Aussage.

Feasibility Precondition (FP): $B_i p \wedge B_i U_j p$

Dies bedeutet im Klartext: Agent *i* kann die Wahrheit von *p* nur dann bestätigen, wenn er selbst glaubt, dass *p* wahr ist und wenn er weiterhin glaubt, dass *j* unsicher über den Wahrheitsgehalt von *p* ist.

Rational Effect (RE): $B_j p$

Klartext: Nach der Aktion soll Agent *j* glauben, dass *p* wahr ist.

9.2. PDB Beispiele

Für jeden Record ist die Bedeutung jeder einzelnen Spalten in der 80 Spalten langen Zeile streng definiert. Als Beispiel sei hier die Definition des „ATOM“ Records detailliert:

Der „ATOM“ Record enthält die Angaben über die Raumposition eines Atoms im Zusammenhang einer Proteinstruktur. Die einzelnen Spalten der Datenzeile werden definiert wie in Tabelle 5 PDB ATOM Record.

ATOM Record Format

COLUMNS	DATA TYPE	FIELD	DEFINITION
1 - 6	Record name	äTOM "	
7 - 11	Integer	serial	Atom serial number.
13 - 16	Atom	name	Atom name.
17	Character	altLoc	Alternate location indicator.
18 - 20	Residue name	resName	Residue name.
22	Character	chainID	Chain identifier.
23 - 26	Integer	resSeq	Residue sequence number.
27	AChar	iCode	Code for insertion of residues.
31 - 38	Real(8.3)	x	Orthogonal coordinates for X in Angstroms.
39 - 46	Real(8.3)	y	Orthogonal coordinates for Y in Angstroms.
47 - 54	Real(8.3)	z	Orthogonal coordinates for Z in Angstroms.
55 - 60	Real(6.2)	occupancy	Occupancy.
61 - 66	Real(6.2)	tempFactor	Temperature factor.
73 - 76	LString(4)	segID	Segment identifier, left-justified.
77 - 78	LString(2)	element	Element symbol, right-justified.
79 - 80	LString(2)	charge	Charge on the atom.

Tabelle 5 PDB ATOM Record

Das folgende Beispiel stellt den ATOM Record im Zusammenhang dar Tabelle 6:

```
         1         2         3         4         5         6         7
8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
ATOM    145  N   VAL A  25      32.433  16.336  57.540  1.00 11.92      A1   N
```

```
ATOM    146  CA  VAL A  25      31.132  16.439  58.160  1.00 11.85      A1   C
ATOM    147  C   VAL A  25      30.447  15.105  58.363  1.00 12.34      A1   C
ATOM    148  O   VAL A  25      29.520  15.059  59.174  1.00 15.65      A1   O
ATOM    149  CB AVAL A  25      30.385  17.437  57.230  0.28 13.88      A1   C
ATOM    150  CB BVAL A  25      30.166  17.399  57.373  0.72 15.41      A1   C
ATOM    151  CG1AVAL A  25      28.870  17.401  57.336  0.28 12.64      A1   C
ATOM    152  CG1BVAL A  25      30.805  18.788  57.449  0.72 15.11      A1   C
ATOM    153  CG2AVAL A  25      30.835  18.826  57.661  0.28 13.58      A1   C
ATOM    154  CG2BVAL A  25      29.909  16.996  55.922  0.72 13.25      A1   C
```

Tabelle 6 ATOM Beispiel

Das PDB Format kennt noch eine Vielzahl weiterer Records, auf die hier aber nicht im Detail eingegangen werden soll:

Titel / Header Bereich:

- HEADER
- OBSLTE
- TITLE
- CAVEAT
- COMPND
- SOURCE
- KEYWDS
- EXPDTA
- AUTHOR
- REVDAT
- SPRSDE
- JRNL
- REMARK
- REMARK 1
- REMARK 2
- REMARK 3
- REMARK 4 – 999

Primär-Struktur-Sektion:

- DBREF
- SEQADV
- SEQRES
- MODRES

Sektion für Heterogene Atome (Atome, die nicht Teil eines Proteins sind, aber dennoch für das Verständnis des im gleichen Datensatz beschriebenen Proteins oder der Nukleinsäure wichtig sind, z.B. eingelagerte Zuckerreste o. Ä.):

- HET
- HETNAM
- HETSYN
- FORMUL

Sekundär-Struktur-Sektion:

- HELIX
- SHEET
- TURN

Sektion für besondere (annotierte) Bindungen:

- SSBOND
- LINK
- HYDBND
- SLTBRG
- CISPEP

Kristallographiesektion:

- CRYST1
- ORIGXn
- SCALEn
- MTRIXn
- TVECT

Koordinatensektion:

- MODEL
- ATOM
- SIGATM
- ANISOU
- SIGUIJ
- TER
- HETATM
- ENDMDL

Bindungen:

- CONECT

Verschiedenes:

- SITE
- MASTER
- END

9.3. SBML Beispiel

Die SBML Notation für das in Abbildung 41 angegebene einfache Reaktionsnetzwerk lautet wie folgt:

```
<?xml version="1.0" encoding=üTF-8"?>
<sbml xmlns="http://www.sbml.org/sbml/level2" level="2" version="1">
      <model id="Branch">
            <notes>
                  <body xmlns="http://www.w3.org/1999/xhtml">
                        <p>Simple branch system.</p>
                        <p>The reaction looks like this:</p>
                        <p>reaction-1: X0 -> S1; k1*X0;</p>
                        <p>reaction-2: S1 -> X1; k2*S1;</p>
                        <p>reaction-3: S1 -> X2; k3*S1;</p>
                  </body>
            </notes>
            <listOfCompartments>
                  <compartment id="compartmentOne" size="1"/>
            </listOfCompartments>
            <listOfSpecies>
                  <species id="S1" initialConcentration="0"
                  compartment="compartmentOne"
                  boundaryCondition="false"/>
                  <species id="X0" initialConcentration="0"
                  compartment="compartmentOne"
                  boundaryCondition="true"/>
                  <species id="X1" initialConcentration="0"
                  compartment="compartmentOne"
                  boundaryCondition="true"/>
                  <species id="X2" initialConcentration="0"
                  compartment="compartmentOne"
                  boundaryCondition="true"/>
            </listOfSpecies>
            <listOfReactions>
                  <reaction id="reaction_1" reversible="false">
                        <listOfReactants>
                              <speciesReference species="X0"/>
                        </listOfReactants>
                        <listOfProducts>
                              <speciesReference species="S1"/>
                        </listOfProducts>
                        <kineticLaw>
                              <math
                              xmlns="http://www.w3.org/1998/Math/MathML
                              ">
                                    <apply>
                                          <times/>
                                          <ci> k1 </ci>
                                          <ci> X0 </ci>
                                    </apply>
                              </math>
                              <listOfParameters>
                                    <parameter id="k1" value="0"/>
                              </listOfParameters>
                        </kineticLaw>
                  </reaction>
                  <reaction id="reaction_2" reversible="false">
                        <listOfReactants>
                              <speciesReference species="S1"/>
```

```
                        </listOfReactants>
                        <listOfProducts>
                              <speciesReference species="X1"/>
                        </listOfProducts>
                        <kineticLaw>
                              <math
                              xmlns="http://www.w3.org/1998/Math/MathML
                              ">
                                    <apply>
                                          <times/>
                                          <ci> k2 </ci>
                                          <ci> S1 </ci>
                                    </apply>
                              </math>
                              <listOfParameters>
                                    <parameter id="k2" value="0"/>
                              </listOfParameters>
                        </kineticLaw>
                  </reaction>
                  <reaction id="reaction_3" reversible="false">
                        <listOfReactants>
                              <speciesReference species="S1"/>
                        </listOfReactants>
                        <listOfProducts>
                              <speciesReference species="X2"/>
                        </listOfProducts>
                        <kineticLaw>
                              <math
                              xmlns="http://www.w3.org/1998/Math/MathML
                              ">
                                    <apply>
                                          <times/>
                                          <ci> k3 </ci>
                                          <ci> S1 </ci>
                                    </apply>
                              </math>
                              <listOfParameters>
                                    <parameter id="k3" value="0"/>
                              </listOfParameters>
                        </kineticLaw>
                  </reaction>
            </listOfReactions>
      </model>
</sbml>
```

Tabelle 7 SBML Beispiel

9.4. CellML Beispiel

An folgendem annotiertem Beispiel wird die Funktionsweise der CellMLverdeutlicht:

```
<model name="hodgkin_huxley_model_excerpt"
      xmlns="http://www.cellml.org/cellml/1.0#"
```

```
      xmlns:cellml="http://www.cellml.org/cellml/1.0#"
      xmlns:cmeta="http://www.cellml.org/metadata/1.0#">

      <!--
      Units definitions which could be referenced from the <variable>
      elements
      would typically be inserted here. Units are discussed in Section
5.
      -->

      <component name="membrane">
            <!-- the following variable is used in other components --
      >
            <variable
                  name="V" initial_value="-75.0"
                  public_interface=öut" units="millivolt" />
            <!-- the following variables are imported from other
            components -->

            <variable name="time" public_interface="in"
            units="millisecond" />
            <variable name="i_Na" public_interface="in"
            units="microA_per_cm2" />
            <variable name="i_K" public_interface="in"
            units="microA_per_cm2" />
            <variable name="i_L" public_interface="in"
            units="microA_per_cm2" />

            <!-- the following variable is only used internally -->
            <variable name="C" initial_value="1.0"
            units="microF_per_cm2" />

            <math xmlns="http://www.w3.org/1998/Math/MathML">
                  <apply id="membrane_voltage_diff_eq"><eq />
                        <apply><diff />
                              <bvar><ci> time </ci></bvar>
                              <ci> V </ci>
                        </apply>
                        <apply><divide />
                              <apply><minus />
                                    <ci> i Na </ci>
                                    <ci> i K </ci>
                                    <ci> i L </ci>
                              </apply>
                              <ci> C </ci>
                        </apply>
                  </apply>
            </math>
      </component>

      <component name="sodium_channel">
            <!-- the following variables are used in other components
      -->
            <variable name="i_Na" public_interface=öut"
            units="microA_per_cm2" />

            <!-- the following variables are imported from other
            components -->
            <variable name="time" public_interface="in"
            units="millisecond" />
```

```
            <variable name="V" public_interface="in" units="millivolt"
      />

            <math xmlns="http://www.w3.org/1998/Math/MathML">
                  <apply id="i_Na_calculation"><eq />
                        <ci> i_Na </ci>
                        ... <!-- a function of V & time -->
                  </apply>
            </math>
      </component>

      <connection>
            <map_components component_1="membrane"
            component_2="sodium_channel" />
            <map_variables variable_1="V" variable_2="V" />
            <map_variables variable_1="i_Na" variable_2="i_Na" />
      </connection>
</model>
```

9.5. BiSCeD Schema

```
<?xml version="1.0" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.c-seiler.de/BiSCeD" xmlns="http://www.c-seiler.de/BiSCeD">
	<xs:simpleType name="positiveFloat">
		<xs:restriction base="xs:float">
			<xs:minInclusive value="0.0" />
		</xs:restriction>
	</xs:simpleType>
	<xs:element name="BiSCeD">
		<xs:complexType>
			<xs:choice>
				<xs:element name="Compound" type="CompoundType" />
				<xs:element name="Space" type="SpaceType" />
				<xs:element name="Field" type="FieldType" />
			</xs:choice>
		</xs:complexType>
	</xs:element>
	<xs:complexType name="CompoundType">
		<xs:choice>
			<xs:element name="ListOfSpecies" type="ListOfSpeciesType" />
			<xs:element name="Put" type="PutType" />
			<xs:element name="Get" type="GetType" />
		</xs:choice>
	</xs:complexType>
	<xs:complexType name="ListOfSpeciesType">
		<xs:sequence minOccurs="1" maxOccurs=„unbound">
			<xs:element name="InitialSpecies">
				<xs:complexType>
					<xs:attribute name="name" type="xs:string"
use="required" />
					<xs:attribute name="compartment" type="xs:string"
use="required" />
					<xs:attribute name="initialAmount"
type="positiveFloat" use="required" />
```

```
                                        <xs:attribute name="boundaryCondition"
type="xs:boolean" use="required" />
                                    </xs:complexType>
                                </xs:element>
                            </xs:sequence>
                        </xs:complexType>
                        <xs:complexType name="PutType">
                            <xs:sequence minOccurs="1" maxOccurs="1">
                                <xs:element name="ListOfVariableSpecies"
type="ListOfVariableSpeciesType" />
                            </xs:sequence>
                        </xs:complexType>
                        <xs:complexType name="GetType">
                            <xs:sequence minOccurs="1" maxOccurs="1">
                                <xs:element name="ListOfVariableSpecies"
type="ListOfVariableSpeciesType" />
                            </xs:sequence>
                        </xs:complexType>
                        <xs:complexType name="ListOfVariableSpeciesType">
                            <xs:sequence minOccurs="1" maxOccurs=„unbound">
                                <xs:element name="Species">
                                    <xs:complexType>
                                        <xs:attribute name="name" type="xs:string"
use="required" />
                                        <xs:attribute name=ämount" type="positiveFloat"
use="required" />
                                    </xs:complexType>
                                </xs:element>
                            </xs:sequence>
                        </xs:complexType>
                        <xs:complexType name="SpaceType">
                            <xs:choice>
                                <xs:element name="SpaceConfiguration"
type="SpaceConfigurationType" />
                                <xs:element name="Intersection" type="IntersectionType" />
                                <xs:element name="IntersectionLine" type="IntersectionLineType" />
```

```
                        <xs:element name="IntersectionResult" type="IntersectionResultType"
/>
                </xs:choice>
        </xs:complexType>
        <xs:complexType name="SpaceConfigurationType">
                <xs:sequence minOccurs="1" maxOccurs=„unbound">
                        <xs:element name="Entitiy" type="EntityType" />
                </xs:sequence>
        </xs:complexType>
        <xs:complexType name="EntityType">
                <xs:sequence minOccurs="1" maxOccurs="1">
                        <xs:element name="Point" type="PointType" />
                        <xs:choice>
                                <xs:element name="BoundingBox" type="BoundingBoxType"
/>
                                <xs:element name="BoundingSphere"
type="BoundingSphereType" />
                        </xs:choice>
                </xs:sequence>
                <xs:attribute name="name" type="xs:ID" use="required" />
        </xs:complexType>
        <xs:attributeGroup name="xyz">
                <xs:attribute name="x" type="xs:float" use="required" />
                <xs:attribute name="y" type="xs:float" use="required" />
                <xs:attribute name="z" type="xs:float" use="required" />
        </xs:attributeGroup>
        <xs:complexType name="PointType">
                <xs:attributeGroup ref="xyz" />
        </xs:complexType>
        <xs:complexType name="BoundingBoxType">
                <xs:attributeGroup ref="xyz" />
        </xs:complexType>
        <xs:complexType name="BoundingSphereType">
                <xs:attribute name="radius" type="positiveFloat" use="required" />
        </xs:complexType>
        <xs:complexType name="IntersectionType">
```

```
            <xs:sequence minOccurs="1" maxOccurs="1">
                  <xs:element name="Point" type="PointType" />
            </xs:sequence>
            <xs:attribute name="id" type="xs:ID" use="required" />
            <xs:attribute name="entity" type="xs:IDREF" use="required" />
      </xs:complexType>
      <xs:complexType name="IntersectionLineType">
            <xs:sequence minOccurs="2" maxOccurs="2">
                  <xs:element name="Point" type="PointType" />
            </xs:sequence>
            <xs:attribute name="id" type="xs:ID" use="required" />
            <xs:attribute name="entity" type="xs:IDREF" use="required" />
      </xs:complexType>

      <xs:complexType name="IntersectionResultType">
            <xs:attribute name="intersection" type="xs:IDREF" use="required" />
            <xs:attribute name="value" type="xs:boolean" use="required" />
      </xs:complexType>
      <xs:complexType name="FieldType">
            <xs:choice>
                  <xs:element name="Singleton" type="SingletonType" />
                  <xs:element name=ärray" type=ärrayType" />
            </xs:choice>
      </xs:complexType>
      <xs:complexType name="SingletonType">
            <xs:attribute name="type" type="xs:NMTOKEN" use="required" />
            <xs:attribute name="value" type="xs:double" use="required" />
            <xs:sequence minOccurs="1" maxOccurs="1">
                  <xs:element name="Point" type="PointType" />
            </xs:sequence>
      </xs:complexType>
      <xs:complexType name=ärray">
            <xs:sequence minOccurs="1" maxOccurs=„unbound">
                  <xs:element name="Element" type="ElementType" />
            </xs:sequence>
```

```
        </xs:complexType>
        <xs:complexType name="ElementType">
                <xs:sequence minOccurs="1" maxOccurs="1">
                        <xs:element name="Vector" type="VectorType" />
                        <xs:element name="Point" type="PointType" />
                </xs:sequence>
                <xs:attribute name="type" type="xs:NMTOKEN" use="required" />
                <xs:attribute name="value" type="xs:double" use="required" />
        </xs:complexType>
        <xs:complexType name="VectorType">
                <xs:attributeGroup ref="xyz" />
        </xs:complexType>
</xs:schema>
```

9.6. BiSCeD DTD

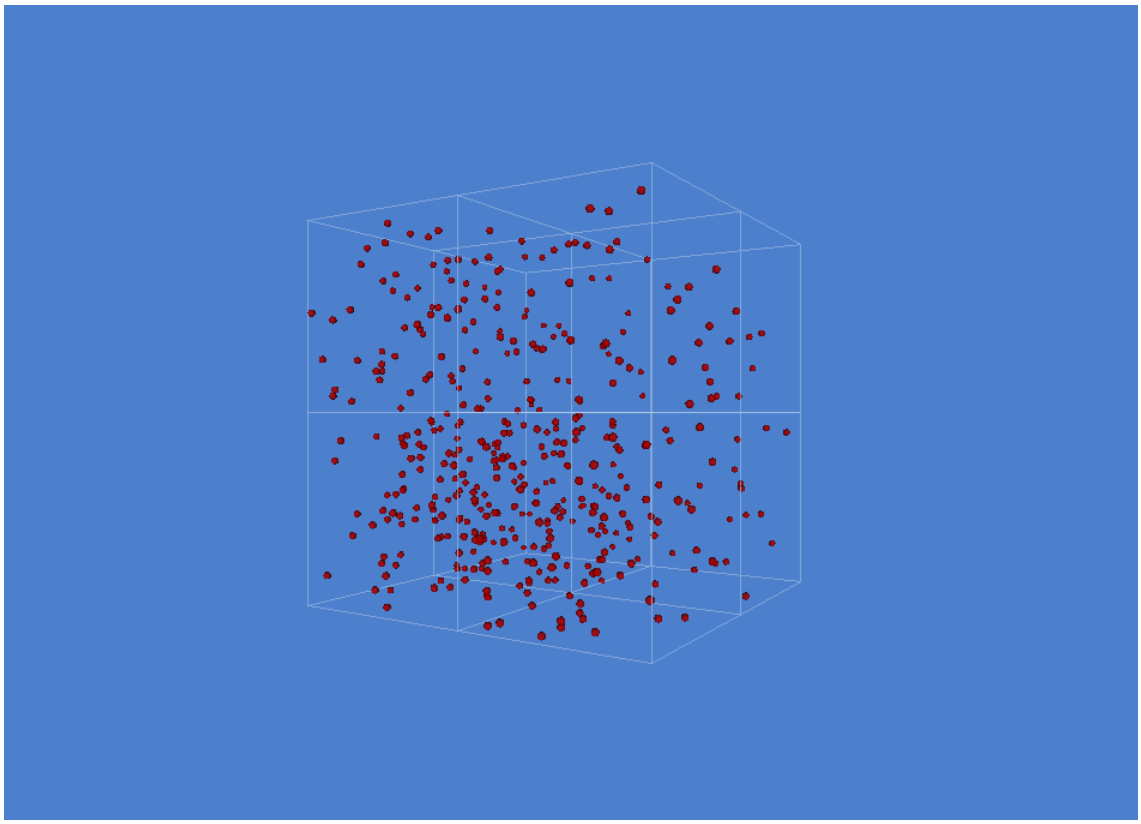
```
<!DOCTYPE  BiSCeD [
<!ELEMENT BiSCeD (Compound | Space | Field)>
]>
<!ELEMENT Compound ( ListOfSpecies | Put | Get)>
<!ELEMENT ListOfSpecies ( InitialSpecies+)>
<!ELEMENT Put ( ListOfVariableSpecies)>
<!ELEMENT Get ( ListOfVariableSpecies)>
<!ELEMENT ListOfVariableSpecies ( Species+)>
<!ELEMENT InitialSpecies  EMPTY>
        <!ATTLIST InitialSpecies
                name CDATA #REQUIRED
                compartment CDATA #REQUIRED
                initialAmount CDATA  #REQUIRED
                boundaryCondition CDATA  #REQUIRED>
<!ELEMENT Species  EMPTY>
        <!ATTLIST Species
                name CDATA #REQUIRED
                amount CDATA  #REQURED>
<!ELEMENT Space ( SpaceConfiguration | Intersection | IntersectionLine |
IntersectionResult)>
<!ELEMENT SpaceConfiguration (Entity+)>
<!ELEMENT Entity (Point, (BoundingBox | BoundingSphere))>
        <!ATTLIST Entity name ID #REQUIRED>
<!ELEMENT Point EMPTY>
        <!ATTLIST Point
                x CDATA #REQUIRED
                y CDATA #REQUIRED
                z CDATA #REQUIRED>
<!ELEMENT BoundingBox EMPTY>
        <!ATTLIST BoundingBox
                x CDATA #REQUIRED
                y CDATA #REQUIRED
                z CDATA #REQUIRED>
```

```
<!ELEMENT BoundingSphere EMPTY>
        <!ATTLIST BoundingSphere
                radius CDATA #REQUIRED>
<!ELEMENT Intersection (Point)>
        <!ATTLIST Intersection
                id ID #REQUIRED
                entity IDREF #REQUIRED>
<!ELEMENT IntersectionLine (Point,Point)>
        <!ATTLIST IntersectionLine
                id ID #REQUIRED
                entity IDREF #REQUIRED>

<!ELEMENT IntersectionResult EMPTY>
        <!ATTLIST IntersectionResult
                intersection IDREF #REQUIRED
                value (True | False) #REQUIRED>
<!ELEMENT Field (Singleton | Array)>
<!ELEMENT Singleton (Point)>
        <!ATTLIST Singleton
                type NMTOKEN #REQUIRED
                value CDATA #REQUIRED>
<!ELEMENT Array (Element+)>
<!ELEMENT Element (Vector,Point)>
        <!ATTLIST Element
                type NMTOKEN #REQUIRED
                value CDATA #REQUIRED>
<!ELEMENT Vector EMPTY>
        <!ATTLIST Vector
                x CDATA #REQUIRED
                y CDATA #REQUIRED
                z CDATA #REQUIRED>
```

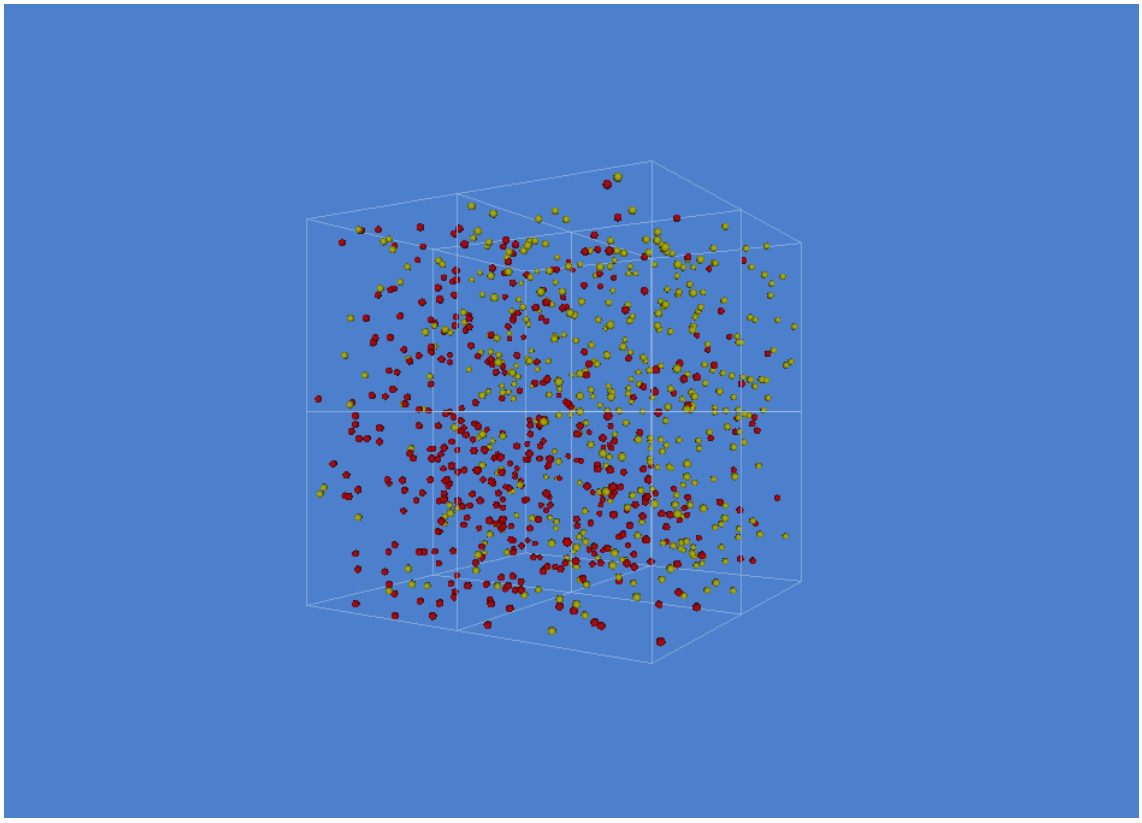
9.7. Exemplarische Ergebnisse des 3D-Reaktions-Diffusionssimulators

Im Folgenden sollen einige exemplarische Ergebnisse dargestellt werden, die mit dem im Rahmen dieser Arbeit entwickelten 3D- Reaktions-Diffusionssimulationssystems erstellt wurden (siehe Kapitel 5.2.3.3) Die Abbildungen wurden zum Teil bereits eingeführt, sie werden hier nur zur besseren Anschauung in vergrößerter Darstellung präsentiert.



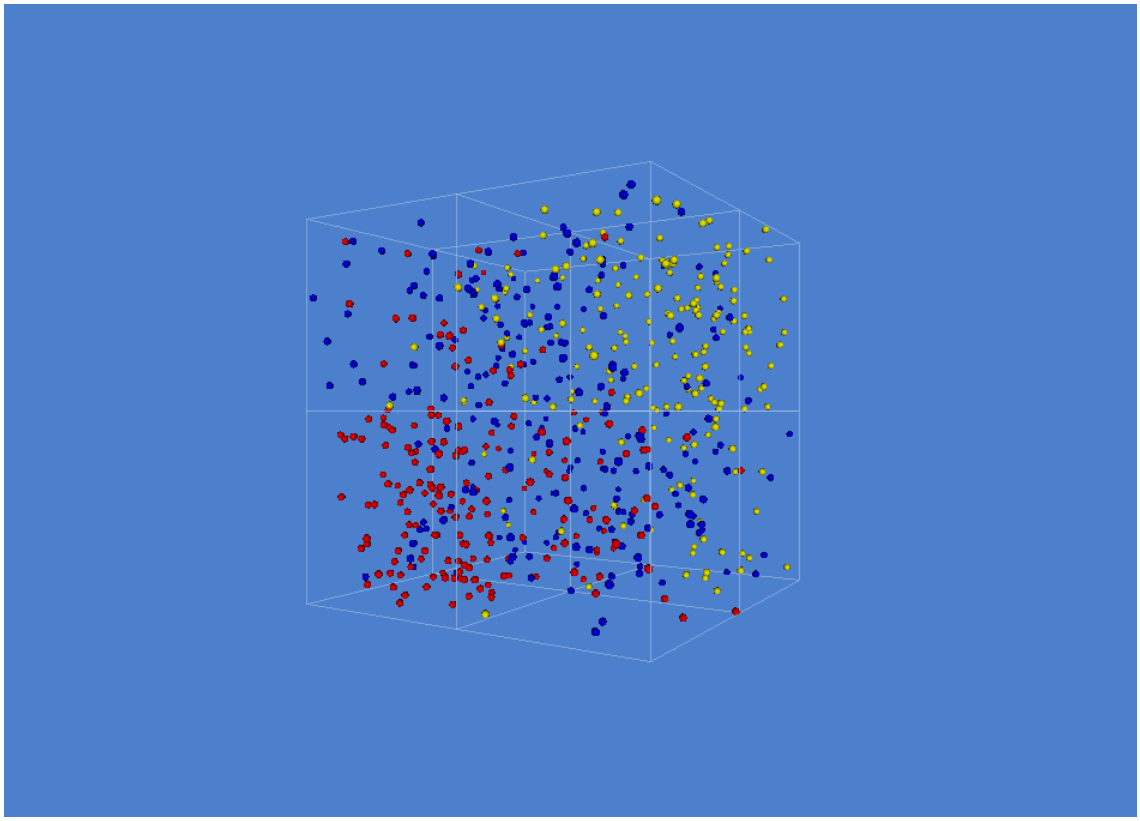
Diese Abbildung zeigt die Diffusion eines Stoffes innerhalb des Volumens.

Deutlich ist zu erkennen, dass sich die maximale Anfangskonzentration vorne, unten rechts befand.



In dieser Abbildung ist die Diffusion zweier Stoffe zu sehen.

Die beiden Stoffe reagiern nicht miteinander.



Ausgehend von zwei Reagenzien, die sich zu Anfang des Experiments vorne, links unten (rot eingfärbt) und hinten, rechts oben (gelb eingefärbt) befanden, wird hier eine Zwischenstufe des Diffusions-Reaktionsexperiments gezeigt.

Das Produkt der Reaktion ist blau eingefärbt.

9.8. Lebenslauf

9.8.1

Christian Seiler

Adolf-Kolping-Str.30
64521 Groß-Gerau

E-mail: c-seiler@gmx.de

* 24. Dezember 1970 in Groß-Gerau

Schule

1990	Abitur, Prälat-Diehl-Schule (Groß-Gerau)

Universität

2000 – 2003	Promotionsstudium, Johann Wolfgang Goethe - Universität Frankfurt am Main.
1991 – 1997	Studium der Informatik: Technische Universität Darmstadt
12/1997	Abschluss: **Diplom-Informatiker**, Diplomarbeit: „Mehrbenutzerunterstützung für VRML 2.0“ gefertigt am Fachgebiet Graphisch-Interaktive Systeme

Berufserfahrung

12/2004 -	Softwareentwickler, InteRes GmbH Darmstadt
07/2004 – 10/2004	Senior-Softwareentwickler, Morgen & Morgen GmbH, Hofheim/Ts.
01/1998 – 12/2003	Angestellter der Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V.
• 07/1999 - 12/2003	Wissenschaftlicher Mitarbeiter, Anwendungszentrum Computergraphik in Chemie und Pharmazie (AGC), Frankfurt / Main
• 01/1999 - 06/1999	Wissenschaftlicher Mitarbeiter, Institut für Graphische Datenverarbeitung (IGD), Darmstadt
• 01/1998 - 12/1998	Wissenschaftliche Hilfskraft mit Abschluss, Institut für Graphische Datenverarbeitung (IGD), Darmstadt
06/1996 - 10/1996	Freier Mitarbeiter CAPCom GmbH (jetzt CAPCom AG)

	Projekt: Aufbau der ersten Internetpräsenz der www.teleauskunft.de der DeTeMedien GmbH
1994 – 1997	Wissenschaftliche Hilfskraft, Fraunhofer-Gesellschaft zur Förderung der angewandten Forschung e.V., Institut für Graphische Datenverarbeitung (IGD), Darmstadt