3D computergraphisches Modellierungswerkzeug für biologische Zellen

Diplomarbeit

von

Jan-Patrick Schäfer

Fraunhofer Anwendungszentrum Computergraphik in Chemie und Pharmazie (AGC), Frankfurt a. M.

Fachhochschule Gießen-Friedberg
Fachbereich Mathematik, Naturwissenschaften und Datenverarbeitung
Studiengang Mathematik

1. Referent:	Prof. Dr.-Ing Dipl.-Math. Monika Lutz
2. Referent:	Prof. Dr. Ralf Dörner
Betreuer des AGC:	Dipl.-Inform. Christian Seiler
	Dipl.-Biol. Jens Barthelmes

Danksagung

An dieser Stelle möchte ich all jenen danken, die mich bei der Erstellung dieser Diplomarbeit fachlich und persönlich unterstützt haben.

Ich bedanke mich bei Frau Prof. Dr.-Ing. Dipl.-Math. Monika Lutz, Herrn Dipl.-Biol. Jens Barthelmes und Herrn Dipl.-Inform. Christian Seiler für die fachliche Betreuung, sowie bei Herrn Prof. Dr. Ralf Dörner für die Übernahme der Korreferentschaft meiner Diplomarbeit.

Ein besonderer Dank gebührt meinen Eltern, die mir mein Studium finanziell ermöglicht haben und mir in jeder schwierigen Situation hilfreich zur Seite standen.

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation und Zielsetzung der Arbeit

Die Biologie befasst sich seit jeher damit, die Prinzipien des Lebens zu ergründen. Die Erforschung der Zelle als Grundbaustein aller lebenden Organismen ist in diesem Zusammenhang eines der Hauptaufgabengebiete der biologischen Forschung.

Das gesamte biologische System der Zelle wird zu diesem Zweck in feinstrukturierte Teilsysteme zerlegt, die einzeln analysiert und beschrieben werden. Das Hauptmittel zur Datengewinnung ist das Laborexperiment. Aufgrund der Verbesserung der Laborverfahren und der Entwicklung von Hochdurchsatzmethoden und -geräten im Zuge des technischen Fortschritts der letzten Jahrzehnte ist es gelungen eine enorme Datenmenge zu produzieren — hier sei insbesondere die fortschreitende Sequenzierung der Genome unterschiedlicher Organismen zu erwähnen.

Dieser Datenbestand birgt die Hoffnung, die einzelnen Teilbereiche des biologischen Gesamtsystems der Zelle umfassend beschreiben und diese in einem ganzheitlichen Ansatz zusammenfassen zu können und somit zu einem grundlegenden Verständnis der Zelle als Ganzem zu gelangen.

Zur Beschreibung eines biologischen Systems wird versucht aus den einzelnen Daten des vorhandenen Datenbestandes Gesetzmäßigkeiten zu erkennen und basierend auf diesen Erkenntnissen mathematische Modelle aufzustellen. Auf Basis dieser Modelle können dann computerbasierte Simulationsverfahren entwickelt werden, die bestimmte Zellprozesse in einer Computersimulation nachbilden.

Die Simulation bietet neue Experimentiermöglichkeiten für Reaktionsvorgänge innerhalb der Zelle. Dadurch lassen sich eventuell zeit- und kostenintensive Lebend[1]- und Reagenzglasexperimente[2] durch computergestützte Experimente[3] ersetzen, was in der medizinischen und der pharmazeutischen Forschung erhebliche Vorteile bringen würde und insbesondere die notwendige Anzahl von Tierversuchen verringern könnte.

[1] in vivo
[2] in vitro
[3] in silicio

Um den Verlauf computersimulierter Zellvorgänge zu visualisieren, müssen bestimmte Benutzerschnittstellen bereitgestellt werden. Bei einem ganzheitlichen Ansatz der Zellsimulation sind unter anderem zellinterne Bewegungsvorgänge im dreidimensionalen Raum zu berücksichtigen. Um solche Bewegungsprozesse in der Simulation darstellen zu können, muss ein dreidimensionales Modell einer Zelle entwickelt werden.

Das Ziel dieser Arbeit ist einerseits die Entwicklung eines Konzeptes für die computerbasierte 3D-Repräsentation einer Zelle in Hinblick auf die Verwendbarkeit dieses Modells für die Simulation von Zellvorgängen. Auf der anderen Seite soll ein Modellierwerkzeug entwickelt werden, das als Hilfmittel zur Modellierung eines solchen repräsentativen Zellmodells eingesetzt werden kann.

1.2 Gliederung der Arbeit

Kapitel 2: Grundlagen

In diesem Kapitel werden Grundlagen zusammengestellt, die für das Verständnis der Arbeit von Bedeutung sind. Neben biologischen Grundlagen über Funktion und Aufbau von Zellen werden computergrafische Grundlagen, UML und einige Design-Patterns vorgestellt.

Kapitel 3: Stand der Technik

Dieses Kapitel versucht einen Überblick über den aktuellen Forschungsstand im Bereich der computerbasierten Zellsimulation zu vermitteln. Außerdem werden die bei der Implementierung des Zelleditor verwendeten Hilfsmittel vorgestellt.

Kapitel 4: Konzept und Implementierung der Klassen für die computerbasierte Repräsentation einer Zelle

Für die Darstellung einer Zelle im Computer wurden mehrere Klassen entwickelt, die in diesem Kapitel vorgestellt werden.

Kapitel 5: Entwicklung der Benutzeroberfläche des Zelleditors

Um eine Zelle modellieren zu könen wurde das Programm CellEdit entwickelt, dessen Aufbau in diesem Kapitel erläutert wird.

Kapitel 6: Implementierung der Klassen zur Modellierung einer Hefezelle

In diesem Kapitel werden, basierend auf den entwickelten Basisklassen, spezialisierte Klassen entwickelt, die dazu verwendet werden können, eine Hefezelle zu modellieren.

Kapitel 7: Evaluierung und Ausblick

Dieses Kapitel enthält eine Bewertung der Ergebnisse der Arbeit und listet Ausblicke auf mögliche zukünftige Erweiterungen zur Optimierung des in dieser Arbeit vorgestellten Zellmodellierungstools auf.

2 Grundlagen

2.1 Biologische Grundlagen

In diesem Abschnitt werden die biologischen Grundlagen über Funktion und Aufbau der Zelle und einiger wichtiger Zellkomponenten behandelt.

Dies soll zum besseren Verständnis der bei der Zellmodellierung auftretenden Anforderungen beitragen und einen Überblick über die räumliche Struktur der Zelle und die damit verbundenen Anforderungen an das Modellierung-Tool geben.

Die allgemeinen Darstellungen von Funktion und Aufbau der Zelle sowie die Beschreibung von Hefezellen sind aus [KlSi99] und [BeGJ01] entnommen. Für die Darstellung von Aufbau und Struktur der Zellkomponenten wurden Informationen aus [UdKo02] verwendet.

2.1.1 Die Zelle

Jeder lebende Organismus besteht aus einer oder mehreren Zellen. Die Zelle ist die kleinste lebens- und vermehrungsfähige Einheit und wird deswegen häufig auch als Elementarorganismus bezeichnet.

Die Zelle muss bestimmte Funktionen abdecken, damit ihr Überleben sichergestellt wird. Eine wichtige Rolle spielen dabei die Informationsmoleküle, sogenannte Genome, die die Aufrechterhaltung der Zellstruktur, die Funktionalität der Zelle und die Selbstreproduktion, den Teilungsprozess der Zelle, steuern.

Sie besitzt die Fähigkeit kontrolliert Stoffe mit ihrer Umwelt auszutauschen. Dieser Stoffwechselprozess dient dazu, die Zelle am Leben zu erhalten. Die Zelle verbraucht Energie durch Stoffaufbau und erzeugt Energie durch Stoffabbau oder durch die Absorption von Licht. Wichtiger Bestandteil des Stoffwechselprozesses ist die jede Zelle umgebende Plasmamembran, die das als Zytoplasma bezeichnete Zellinnere von der Außenwelt abgrenzt und zusätzlich einen kontrollierten Stoffaustausch mit der Umwelt ermöglicht.

Zellen können bestimmte chemische oder physikalische Reize empfangen und entsprechend auf diese Reize reagieren, eine in vielen Fällen weitere wichtige Funktion von Zellen ist die Möglichkeit zur Bewegung.

2.1.2 Klassifizierung von Zellen

Es wird allgemein zwischen zwei verschiedenen Grundformen der Zellorganisation unterschieden.

Die erste Organisationsform ist die Protocyte, die Zelle der einzelligen Prokaryoten, der bakteriellen Organismen. Die Prokaryoten werden in zwei Reiche untergliedert, das Reich der Eubakterien, das den größten Teil der bekannten Bakterienarten umfasst und das Reich der meist in extremen Lebensräumen lebenden Archaebakterien.

Protocyten besitzen in ihrem Inneren in der Regel nur einen Reaktionsraum (Kompartiment), der durch die Plasmamembran begrenzt wird. Die Erbinformationen dieser Zellen befinden sich im Zytoplasma. Viele Prokaryoten besitzen Geißel, die dem Organismus zur Fortbewegung dienen.

Die zweite Organisationsform ist die Eucyte, die Zelle der Eukaryoten, deren Gruppe alle Organismen umfasst, die nicht den Prokaryoten zugeordnet werden. Die Eucyte ist wesentlich komplexer aufgebaut als die Protocyte. Insbesondere befindet sich die DNA in einem membranbegrenzten Bereich innerhalb der Zelle, dem Zellkern. Neben dem Zellkern besitzt die Eucyte noch weitere membranbegrenzte Untereinheiten, die innerhalb der Zelle eigene Kompartimente bilden. Außerdem enthält sie semiautonome[1] Organelle, wie z.B. die Mitochondrien oder die Plastiden der Pflanzenzellen.

Unter den Eukaryoten gibt es einzellige und mehrzellige Organismen. Mehrzellige Organismen sind aus unterschiedlichen spezialisierten Zelltypen aufgebaut. Diese spezialisierten Zellen decken jeweils einen bestimmten Aufgabenbereich im Lebensprozess des Organismus ab. Das Überleben des Organismus wird durch die Kooperation aller unterschiedlichen Zelltypen ermöglicht. Spezialisierte Zelltypen des menschlichen Organismus sind z. B. Muskelzellen, Nervenzellen oder Blutzellen.

Die Eukaryoten lassen sich in weitere Reiche untergliedern. In [KlSi99] wird eine Unterteilung in die Reiche der Archaezoen, Protozoen, Pilze, Pflanzen und Tiere durchgeführt. Das Reich der Archaezoen umfasst nur wenige einzellige Organismen, die zumeist parasitisch in Vielzellern leben, Protozoen sind einzellige Organismen, die eine komplexe innere Struktur aufweisen. Die übrigen drei Reiche beinhalten größtenteils mehrzellige Organismen, wobei auch einzellige Organismen zu finden sind (z. B. einige Algen und Hefen).

Die Klassifikation von Organismen kann durch weitere Hierarchieebenen ergänzt werden. Üblicherweise wird eine weiterführende Klassifizieren durch Zuteilung der Organismen in die Kategorienränge Stamm (Phylum), Klasse (Classis), Ordnung (Ordo), Familie (Familia), Gattung (Genus) und Art (Species) durchgeführt (siehe Abb. 2.1).

2.1.3 Aufbau und Struktur einer Zelle

Eukaryotische und Prokaryotische Zellen sind sehr unterschiedlich aufgebaut. In eukaryotischen Zellen befinden sich sehr viel komplexere Strukturen, verschiedene Kompartimente und sogar semiautonome Organelle, während prokaryotischen Zellen häufig nur einen einzigen Reaktionsraum besitzen. Da bei der Entwicklung von 3D-Zellmodellen der Schwerpunkt auf der Entwicklung von eukaryotischen Zellmodellen

[1] semiautonome Organelle verfügen über eigene Erbinformationen, die allerdings nicht ausreichen um alle vom Organell benötigten Proteine zu codieren

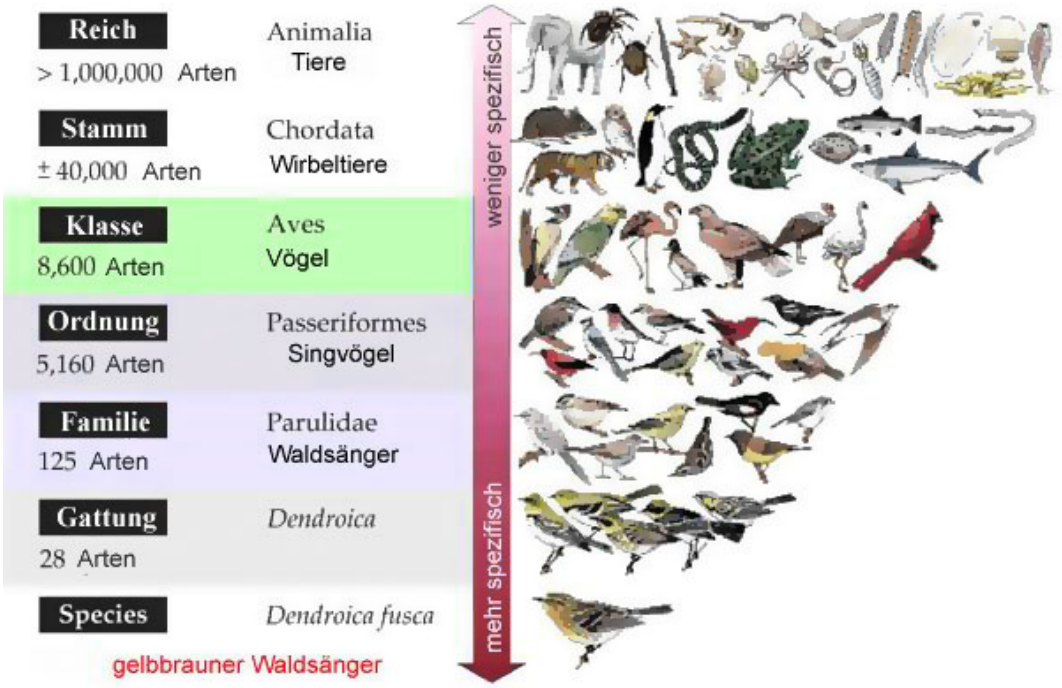

Abbildung 2.1: Klassifizierung eines gelbbraunen Waldsängers [Biol]

liegt, wird im Folgenden lediglich die Struktur von eukaryotischen Zellen genauer dargestellt.

2.1.3.1 Eukaryotische Zellen

Jede Eukaryotische Zelle wird von einer Plasmamembran begrenzt. Im Inneren der Zelle, dem Zytoplasma, befinden sich neben chemischen Stoffen (Wasser, Salze, Proteine) und toten Zelleinschlüssen (Fette, Pigmente) mehrere als Zellorganelle bezeichnete Strukturen.

In [Broc01] werden Zellorganelle als „in Zellen vorkommende organähnliche Gebilde mit spezifischer Struktur und Funktion“ beschrieben. Als Organelle werden dort unter anderem Zellkern, Mitochondrien, Plastiden, der Golgi-Apparat, das Endoplasmatische Retikulum, Ribosomen und das Zytoskelett bezeichnet.

Eine normale eukaryotische Zelle besitzt normalerweise einen Zellkern, ein Endoplasmatisches Retikulum, ein Zytoskelett, sowie einen Golgi-Apparat und mehrere Mitochondrien (siehe Abb. 2.2).

Pflanzen- und Pilzzellen besitzen zusätzlich zur Plasmamembran noch eine stabile äußere Begrenzung, eine Zellwand und durch eine Membran abgegrenzte flüssigkeitgefüllte Bereiche, sogenannte Vakuolen. In Pflanzenzellen treten außerdem verschiedene Plastiden auf (siehe Abb. 2.3).

Die einzelnen Komponenten der Zelle werden nun etwas näher betrachtet.

2.1.3.2 Zellkern

Der Zellkern (Nucleus) ist der Hauptträger der Erbinformationen der Zelle. Er ist durch eine doppelte Membran nach außen abgegrenzt, die einen Zwischenraum, die

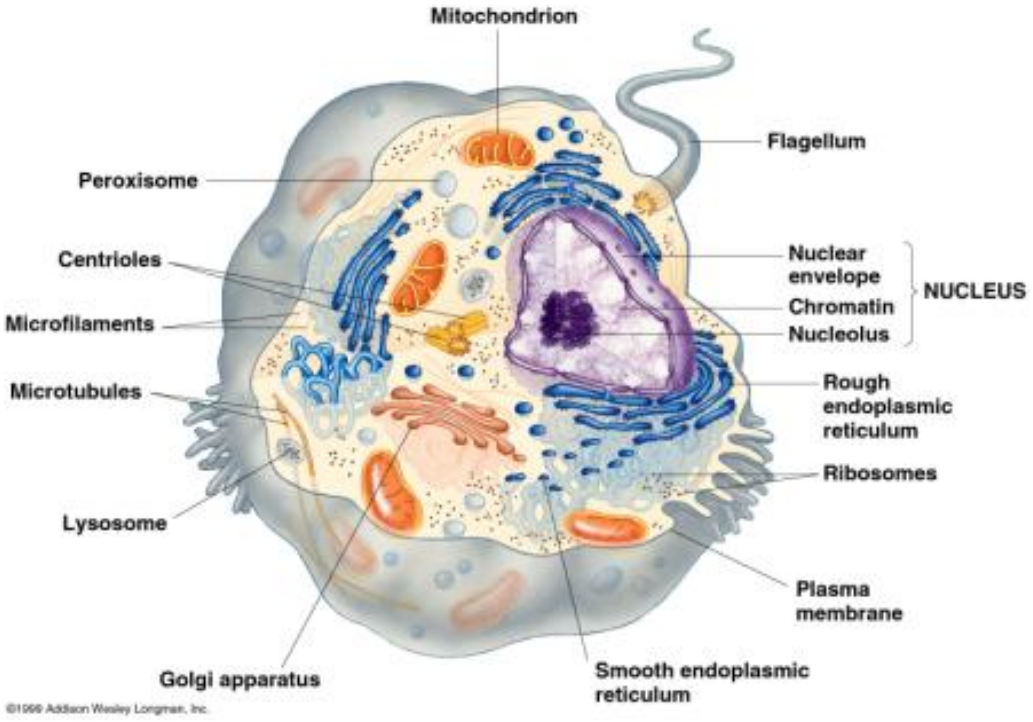

Abbildung 2.2: Darstellung einer Pseudo-Tierzelle [Faje]

sogenannte perinucleäre Zisterne bildet. Verbindungen nach außen bestehen durch die Kernporen, die in größerer Anzahl auf der Kernhülle vorhanden sind. Die perinucleäre Zisterne ist mit dem Endoplasmatischen Retikulum verbunden (siehe Abb. 2.4 und Abb. 2.5).

Im Inneren des Zellkerns befindet sich ein aus den Chromosomen aufgebautes Chromatingerüst, das die DNA enthält und bei der Zellteilung eine entscheidende Rolle spielt sowie ein oder mehrere Kernkörperchen (Nucleoli), die Ribonucleoproteidpartikel produzieren, die Bausteine der Ribosomen. Die Anzahl der Chromosomen variiert bei den verschiedenen Organismen, ist aber bei unterschiedlichen Zelltypen eines Organismus immer gleich.

Die Hauptfunktion des Kernes ist die Steuerung verschiedener Zellprozessen, insbesondere der Proteinsynthese und der Zellteilung. Die Größe und die Form des Kerns ist bei den verschiedenen Zellformen unterschiedlich ausgeprägt und hängt vom Aktivitätszustand der Zelle ab, nimmt oftmals aber ungefähr 10% des Volumens der Zelle ein. Die Größe des Kerns kann von 0,5 μm bei einigen Pilzen bis zu 0,5 mm bei Eizellen einiger nacktsamiger Pflanzen variieren, ist normalerweise aber nicht größer als 5 μm. Der Zellkern ist bei den meisten Zelltypen in der normalen Phase ein kugelförmiges oder ovoides Gebilde, das sich zumeist in einer zentralen Region der Zelle befindet.

2.1.3.3 Mitochondrien

Mitochondrien sind semiautonome Organellen der eukaryotischen Zelle, die eine entscheidende Funktion bei der Energieversorgung der Zelle einnehmen. Sie besitzen eine Doppelmembran, deren innerer Teil starke Einbuchtungen in den Innenraum

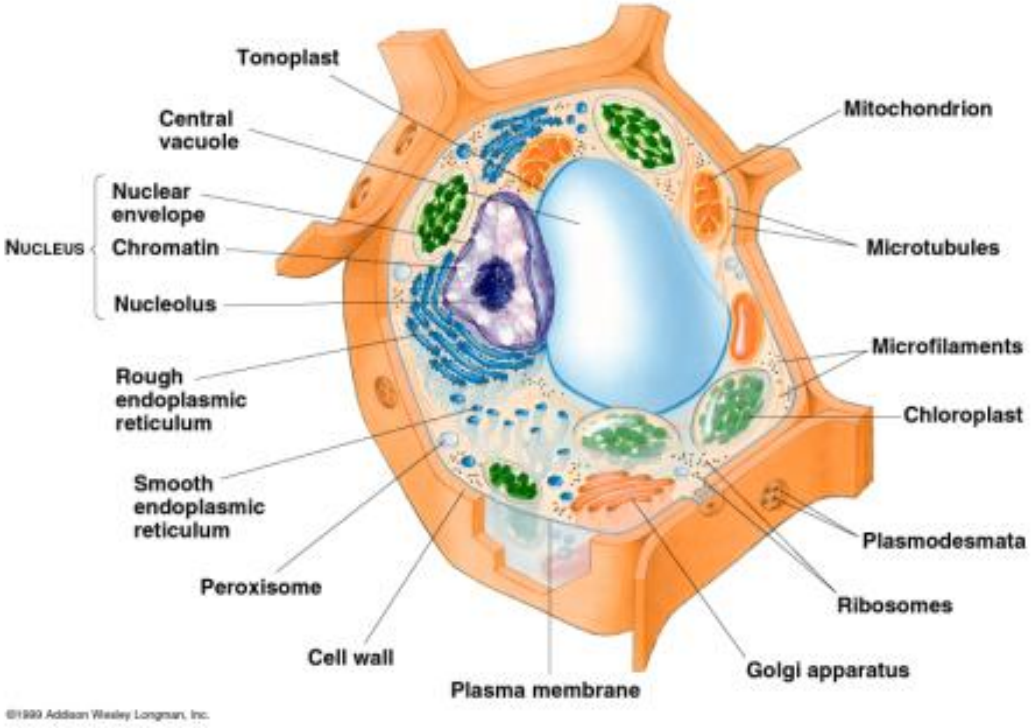

Abbildung 2.3: Darstellung einer Pseudo-Pflanzenzelle [Faje]

vorweist. Fingerförmige Einkerbungen werden als Tubuli, scheibenförmige als Cristae bezeichnet (siehe Abb. 2.6 und Abb. 2.7).

Mitochondrien treten in unterschiedlicher Anzahl in verschiedenen Zelltypen auf. Es besteht oftmals ein Zusammenhang zwischen der Aktivität einer Zelle und deren Mitochondrienzahl. Aktive Zellen enthalten eine größere Anzahl von Mitochondrien als weniger aktive.

Von der Form her sind Mitochondrien lang gestreckt oder sphärisch-ovoid ausgebildet. Im Durchschnitt besitzen sie einen Durchmesser von 0,5-1 μm und eine Länge von 1-6 μm.

2.1.3.4 Plastiden

Plastiden sind semiautonome Organellen, die nur in Pflanzenzellen zu finden sind. Sie besitzen eine innere und eine äußere Membran. In ihrem Inneren befinden sich flache Membransäckchen, sogenannte Thylakoide (siehe Abb. 2.9).

Es gibt verschieden Arten von Plastiden, die nach den in ihrem Inneren vorkommenden Pigmenten als Chloroplasten, Chromoplasten und Leukoplasten bezeichnet werden.

Der wichtigste Plastidentyp ist der Chloroplast, der seinen Namen durch den Besitz großer Mengen von Chlorophyll trägt und in den grünen Pflanzenzellen für den Photosyntheseprozess, der Erzeugung von Energie durch die Absorption von Sonnenlicht, verantwortlich ist.

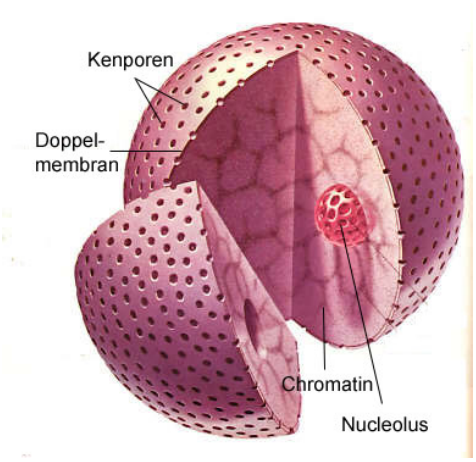

Abbildung 2.4: Schematische Darstellung eines Zellkerns [BevS] (geändert)

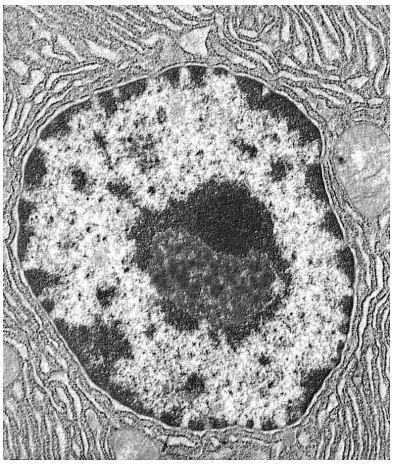

Abbildung 2.5: Elektronenmikroskopische Aufnahme eines Zellkerns [Bart]

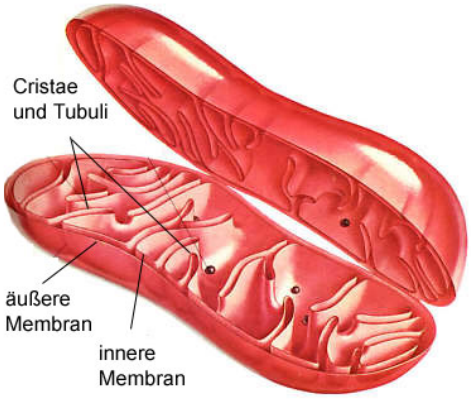

Abbildung 2.6: Schematische Darstellung eines Mitochondriums [BevS] (geändert)

Abbildung 2.7: Elektronenmikroskopische Aufnahme eines Mitochondriums [Ross]

2.1.3.5 Ribosomen

Ribosomen sind aus zwei Ribonucleoproteid-Untereinheiten aufgebaut und ca. 18-20 nm groß. Es gibt zwei Arten von Ribosomen. Der 70S[2]-Typ ist in Protocyten, aber auch in Mitochondrien und Plastiden zu finden, der 80S-Typ befindet sich in eukaryotischen Zellen (siehe Abb. 2.8).

Die Untereinheiten der Ribosomen werden im Kernkörperchen des Zellkerns synthetisiert und vereinigen sich nach Verlassen des Kerns. Ribosomen treten als freie Ribosomen im Kernplasma auf, sind aber auch auf einigen Membranen angesiedelt, wie zum Beispiel auf der äußeren Membran des Zellkerns oder auf den membranbegrenzten Zisternen des granulären Endoplasmatischen Retikulums. Sie spielen eine entscheidende Rolle bei der Synthese von Proteinen.

[2]S steht für Svedberg und ist die Bezeichnung der Maßeinheit der Sedimentationsgeschwindigkeit eines Moleküls in einer Ultrazentrifuge

Abbildung 2.8: Schematische Darstellung eines Ribosoms [UdKo02] (geändert)

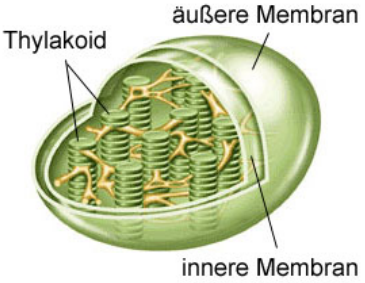

Abbildung 2.9: Schematische Darstellung eines Plastiden [MaC01] (geändert)

2.1.3.6 Endoplasmatisches Retikulum

Als Endoplasmatisches Retikulum (ER) wird ein System aus membranbegrenzten Zisternen bezeichnet, das sich über die ganze Zelle erstreckt und mit der perinucleäre Zisterne des Zellkerns verbunden ist.

Es wird zwischen zwei verschiedene ER-Typen unterschieden. Das granuläre Endoplasmatische Retikulum (GER) besteht in der Regel aus flachen, spaltförmigen Zisternen und Zisternenstapeln, die oftmals parallel zueinander verlaufen, aber auch wirbelartige Strukturen bilden. Auf der Membranaußenseite lagern sich Ribosomen an. Das agranuläre (glatte) Endoplasmatische Retikulum (SER) besteht aus einem System vielfältig verzweigter tubulärer Strukturen. Die agranuläre Form enthält keine Ribosomen auf den Außenmembranen (siehe Abb. 2.10).

Die meisten Zellen besitzen ausschließlich ein granuläres ER, glattes ER ist nur in einigen wenigen spezialisierten Zellen zu finden. Die beiden Formen können innerhalb solcher Zellen kontinuierlich ineinander übergehen.

Das Endoplasmatische Retikulum ist ein zellinternes Transportsystem. Die granuläre Form hat aufgrund der auf den Membranaußenseiten angelagerten Ribosomen einen wichtigen Anteil bei der Proteinsynthese.

2.1.3.7 Golgi-Apparat

Das Grundelement des Golgi-Apparates ist die Golgi-Zisterne, ein flaches Membranvesikel von 1-2 μm Durchmesser. Mehrere solcher Zisternen bilden Funktionseinheiten, sogenannte Dictyosomen. Die Gesamtheit aller Dictyosomen der Zelle wird als Golgi-Apparat bezeichnet. Die Anzahl von Dictyosomen kann in den unterschiedlichen Zellformen von einem bis über hundert schwanken.

Die Zisternen der Dictyosomen schnüren kleine Membranvesikel ab, die für den Transport von Stoffen verwendet werden. Die Elemente des Golgi-Apparates stehen über solche Transportvesikel in Verbindung mit dem Endoplasmatischen Retikulum und der Plasmamembran der Zelle (siehe Abb. 2.11).

2.1.3.8 Vakuole

Als Vakuole wird ein membranbegrenzter flüssigkeitsgefüllter Hohlraum im Inneren von Zellen bezeichnet. Vakuolen sind in den meisten Pilz- und Pflanzenzellen zu

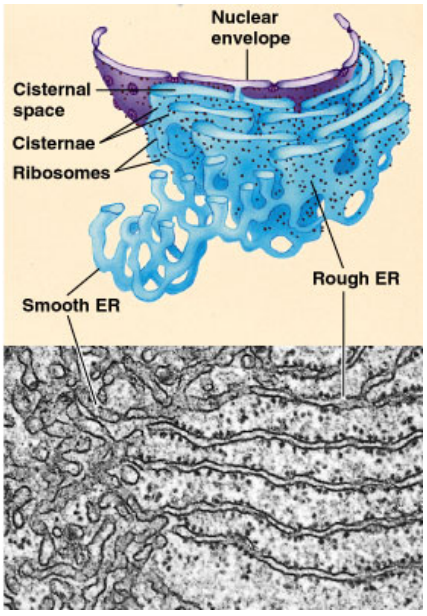

Abbildung 2.10: Schematische Darstellung und Elektronenmikroskopische Aufnahme des Endoplasmatischen Retikulums; [Faje] (geändert)

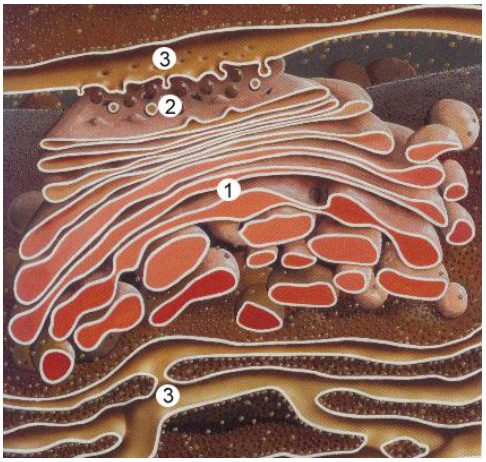

Abbildung 2.11: Schematische Darstellung eines Dictyosoms; *1 - Zisternenstapel, 2 - Transportvesikel, 3 - Endoplasmatisches Retikulum* [UdKo02] (geändert)

finden. Oftmals ist nur eine zentrale Vakuole vorhanden, die allerdings einen Großteil des Zellinnenraums ausfüllt. Die Vakuole kann in ausgewachsenen Zellen bis zu 90% des Volumens der Zelle einnehmen. Vakuolen besitzen eine Speicherfunktion und dienen der Druckstabilisierung der Zelle sowie dem Stoffabbau.

2.1.3.9 Zytoskelett

Die innere Struktur und die Gestalt der Zelle wird durch ein Netzwerk von Proteinfilamenten, dem Zytoskelett, bestimmt, das sich über die ganze Zelle ausstreckt und diese dadurch stabilisiert und festigt. Die fadenförmigen Filamentstrukturen des Zytoskeletts dienen außerdem als Leitbahnen für den zielgerichteten zellinternen Transport von Stoffen (siehe Abb. 2.12).

Es gibt unterschiedliche Filamenttypen, die das Zytoskelett bilden. Diese Filamente entstehen aus Proteinen und bilden fadenförmige Strukturen aus. In [KlSi99] werden insbesondere drei Gruppen von Proteinfilamenten dargestellt:

Intermediärfilamente

Intermediärfilamente, auch als 10nm-Filamente bezeichnet, sind in den meisten Tierzellen vorhanden, in Pflanzenzellen ist ihr Auftreten allerdings noch nicht nachgewiesen worden. Sie geben der Zelle Form und Festigkeit und besitzen einen Durchmesser von 7-12 nm.

Actinfilamente

Actinfilamente besitzen einen Durchmesser von 6 nm. Ihre Länge kann von 0,06 μm bis zu mehr als 1 mm variieren. Actinfilamenten sind in nahezu allen Eucyten zu finden. Sie bilden meist parallele Bündel aus mehreren Einzelfilamenten. Actinfilamente weden auch als Mikrofilamente bezeichnet.

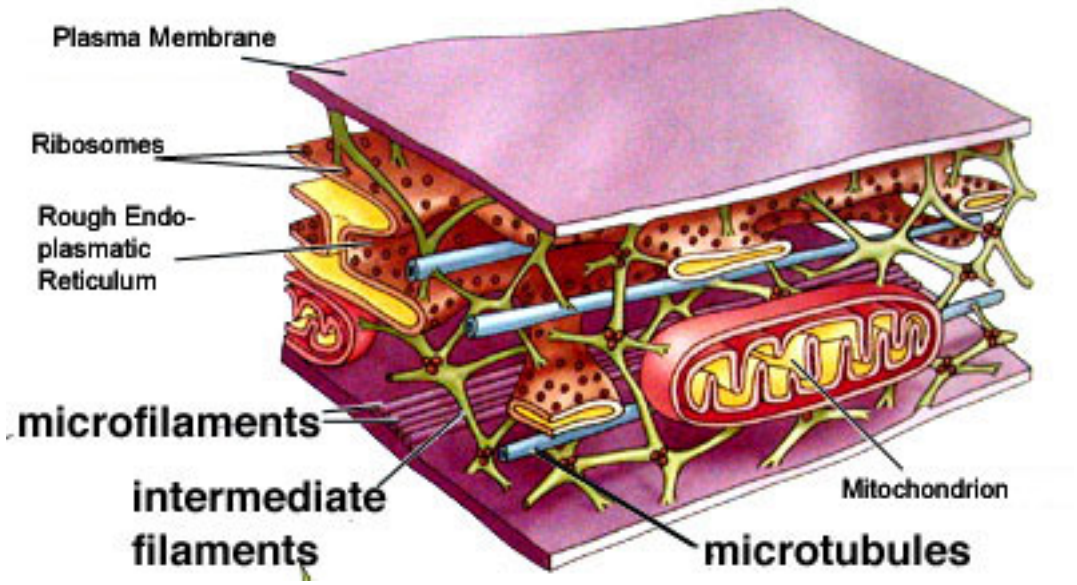

Abbildung 2.12: Zytoskelett einer Zelle [Nish03] (geändert)

Mikrotubuli

Wie Actinfilamente sind auch Mikrotubuli in fast allen Eucyten vorhanden. Mikrotubuli sind röhrenförmige Strukturen mit einem Außendurchmesser von 25 nm und einem Innendurchmesser von 15 nm. Sie können bis zu 100 μm lang werden.

Bei der Bildung von Mikrotubuli nimmt das Zentralkörperchen, das Zentriol, eine entscheidende Stellung ein. Das Zentriol besteht selbst aus neun Tripletts von Mikrotubuli-Elementen, die zusammen einen 400 nm langen und 200 nm durchmessenden zylinderförmigen Komplex bilden (siehe Abb. 2.13). Zentriolen treten meistens als senkrecht zueinander stehendes Zentriolenpaar, dem sogenannten Diplosom, auf. Das Diplosom ist als MTOC (Microtubule Organizing Center) für die Organisation der Bildung der Mikrotubuli verantwortlich. Die Mikrotubuli breiten sich von ihrem Entstehungszentrum sternförmig im Innenraum der Zelle aus.

2.1.4 Hefezellen

Hefezellen sind mikroskopisch kleine Pilze, die vorwiegend als einzellige Organismen auftreten und sich mit einigen Ausnahmen durch Knospung vermehren. Hefen haben eine wirtschaftliche Bedeutung in der Nahrungs- und Genussmittelindustrie. Da Hefezellen viele Komponenten besitzen, die auch in höheren mehrzelligen eukaryotischen Zellen vorhanden sind und es einfach ist, Kulturen von Hefezellen zu produzieren, dienen sie häufig als Modellsystem eukaryotischer Zellen bei molekularbiologischen Arbeiten. Die Zellen der Saccharomyces cerevisiae (Bierhefe) waren die ersten Eukaryoten, deren Genom vollständig sequenziert wurde (1997).

Der Aufbau einer Hefezelle wird am Beispiel der Bierhefe (S. cerevisiae) nun anhand der Informationen aus [Feld01] dargestellt (siehe Abb. 2.14).

Bierhefezellen besitzen normalerweise eine ellipsoidische Form. Der große Durchmesser beträgt in der Regel 5 bis 10 μm, der kleine 1 bis 7 μm.

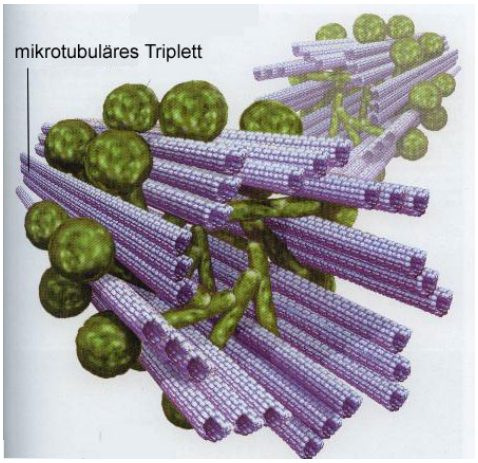

Abbildung 2.13: Schematische Darstellung eines Zentriols [UdKo02] (geändert)

Die Organelle und Strukturen, die in Hefezellen identifiziert werden können, werden nun kurz beschrieben:

Zellwand und Plasmamembran: Neben einer die Zelle umschließenden Plasmamembran besitzt die Hefezelle zusätzlich eine Zellwand als äußerste Hülle. Die Plasmamembran ist ca 7 nm dick, die Zellwand ist mit 100 bis 200 nm wesentlich dicker gebaut. Zwischen Plasmamembran und Zellwand befindet sich ein sehr schmaler Zwischenraum (35-45 A), das sogenannte Periplasma.

Zellkern: Der Zellkern der Hefezelle besitzt einen Durchmesser von ca. 1,5 μm. Der Zellkern wird von einer Doppelmembran begrenzt, auf der sich Zellporen mit einem Durchmesser von 50-100 nm befinden. Die Hefezelle besitzt 16 Chromosomen.

Mitochondrien: Die Mitochondrien der Hefezelle sind dynamische Strukturen und können in Abhängigkeit von der Zellphase und der Belastung der Zelle in Anzahl, Größe und Form stark variieren,

Endoplasmatisches Retikulum: Hefezellen besitzen nur rauhes ER, auf dessen Außenmembranen Ribosomen abgelagert sind.

Golgi-Apparat: Der Golgi-Apparat dieser Hefezelle besteht nur aus einem einzigen Dictyosom.

Zytoskelett: Hefezellen besitzen ein strukturgebendes Zytoskelett aus Proteinfilamenten.

Vakuole: Die Hefezelle besitzt eine zentrale Vakuole, die den Großteil des Zellinnenraums einnimmt.

Sprossungsnarbe: Sprossungsnarben sind ringförmige konvexe Vorsprünge auf der Oberfläche der Zelle, die nach der Knospung einer Kindzelle bei der Mutterzelle zurückbleiben.

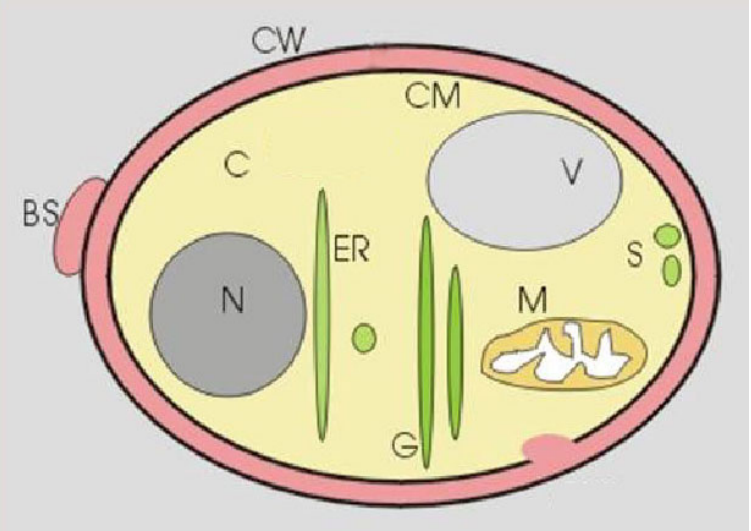

Abbildung 2.14: Schematische Darstellung einer Hefezelle; *C - Zytoplasma, CM - Plasmamembran, CW - Zellwand, N - Nucleus, M - Mitochondrium, V - Vakuole, G - Golgi-Apparat, ER - Endoplasmatisches Retikulum, S - Vesikel, BS - Sprossungsnarbe* [Feld01] (geändert)

2.2 Grundlagen der 3D-Darstellung

2.2.1 3D-Repräsentationsmethoden

Um ein dreidimensionales Objekte in einem 3D-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 [Schi02]. Zwei dieser Methoden, die Repräsentation durch Elementarobjekte und die Repräsentation über Flächenmodelle, wurden bei der 3D-Darstellung der Zelle verwendet und werden nun kurz erläutert.

2.2.1.1 Elementarobjekte

Objekte können in ihrem Aussehen häufig hinreichend genau genug durch das Zusammensetzen unterschiedlicher als Elementarobjekte [VoMü00] oder vordefinierte Primitive [Schi02] bezeichneter 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 2.15 ist ein aus Elementarobjekten modelliertes Modell eines Schreibtischstuhls dargestellt.

2.2.1.2 Flächenmodell

Bei der Flächenmodelldarstellung wird das Objekt durch eine Kombination aus analytischen oder approximierten Flächen (Bezier- oder Splineflächen) beschrieben

[VoMü00]. In der computergrafischen Darstellung werden diese Flächen durch konvexe Polygone nachgebildet. Jedes dieser Polygone wird durch Eckpunkte definiert, die durch Kanten miteinander verbunden werden. Die Darstellung der Kantenzüge ohne Zeichnen der Flächen wird als Drahtgitterdarstellung bezeichnet (siehe Abb. 2.16).

Abbildung 2.15: Ein aus Quadern, Kugeln und einem Zylinder zusammengesetztes Modell eines Schreibtischstuhls

Abbildung 2.16: Drahtgitter- und Flächenmodelldarstellung eines Keramikgefäßes [pVHFr] (geändert)

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.

2.2.2 Koordinatentransformationen

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

Jede der hintereinandergeschalteten Matrizenen ist für eine bestimmte Koordinatentransformation zuständig. In Abbildung 2.17 ist die Koordinatentransformations-Pipeline für eine Rendering-Operation einer Szene skizziert. Jede Transformation bildet die Koordinaten eines bestimmten Koordinatensystem auf ein anderes ab. Die in der Abbildung dargestellten unterschiedlichen Koordinatensysteme werden nun kurz erläutert (siehe dazu [JrSw99]):

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 Model Matrix 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.

Kamera-Koordinaten: Um ein Bild der Szene auf eine Fläche projizierern 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 Kamera-Koordinatensystem 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 dieser Methoden werden nun kurz erläutert (siehe Abb. 2.18) [Schi02]:

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 vewendet.

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.

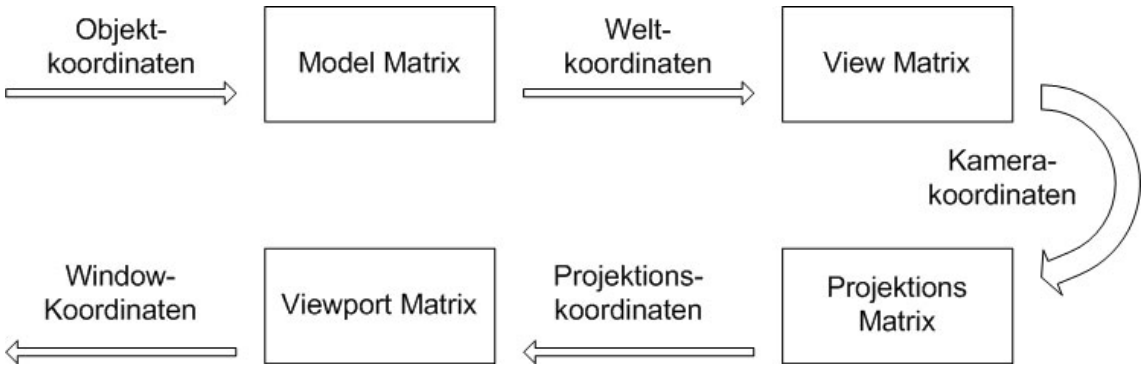

Abbildung 2.17: Koordinatentransformations-Pipeline

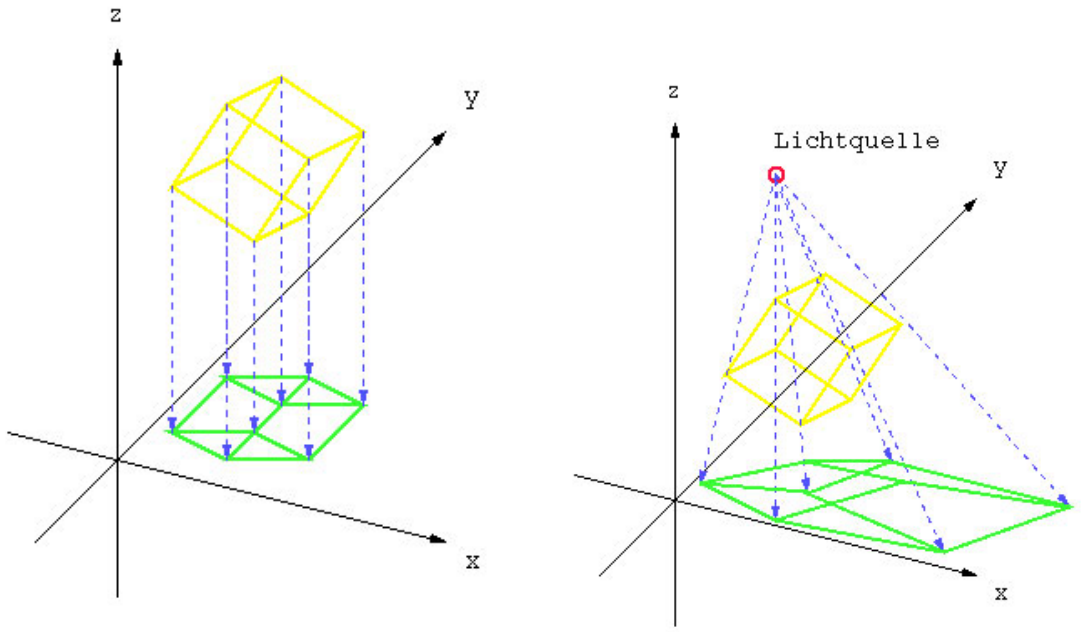

Abbildung 2.18: Darstellung einer orthogonalen (links) und einer perspektivischen (rechts) Projektion [Schi]

2.2.3 Szenengraphen

Eine hierarchisch angeordnete dreidimensionale Szene lässt sich durch einen gerichteten azyklischen Baum, einen sogenannten Szenengraphen, darstellen [Ende99].

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 Abb. 2.19).

Unterschiedliche Szenengraph-Modelle können verschiedene Knotentypen besitzen. Es wird nun eine kurze Beschreibung von allgemein üblichen Knotentypen aufgeführt [Hitc]:

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.

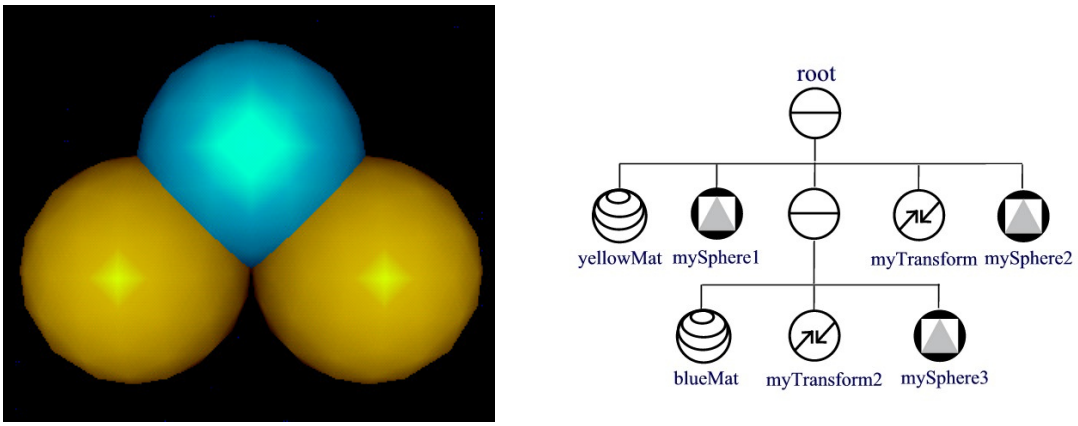

Abbildung 2.19: Szenengraph (rechts) und die dazugehörige Darstellung einer Szene [SVis]

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 [Osfia] 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 gleichzeitiges 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 Auteilung 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. Unter C++ gibt es mehrere OpenGL-basierte Szenengraph-APIs, wie OpenGL Performer, OpenSG oder OpenSceneGraph.

2.3 UML und Design Patterns

2.3.1 UML

Die Unified Modeling Language (UML) [UML] ist eine standardisierte Modellierungssprache, die in der objektorientierten Softwareentwicklung eingesetzt wird , um den Aufbau des Systems zu visualisieren, spezifizieren, konstruieren und dokumentieren.

Sie bietet umfangreiche Möglichkeiten die Struktur von Softwareprojekten mit Hilfe verschiedener Diagramme zu beschreiben und ermöglicht somit Baupläne der Projekte zu erstellen, die von den an der Entwicklung beteiligten Personen gelesen und verstanden werden können.

Es werden nun verschiedene Diagrammtypen vorgestellt, die durch die UML bereitgestellt werden:

Use-Case-Diagramme beschreiben das System in Hinblick auf Benutzer und deren Handlungen und Interaktionen mit dem System.

Klassendiagramme stellen die statischen Beziehungen zwischen Klassen, bzw. zwischen den Objekten der Klassen dar.

Interaktionsdiagramme beschreiben zeitliche Abläufe bestimmter Aktionen.

Package-Diagramme dienen zur Strukturierung der Elemente des Gesamtsystems in unterschiedliche Packages.

Zustandsdiagramme beschreiben das dynamische Verhalten einzelner Klassen in Hinblick auf ihren internen Zustand.

Aktivitätsdiagramme sind Diagramme, die parallel verlaufende Prozesse beschreiben können.

Implementierungsdiagramme stellen Zusammenhänge des Implementierungsgerüstes auf Soft- und Hardwarebasis dar.

In dieser Arbeit werden Klassendiagramme zum Darstellen der entwickelten Klassenstruktur und ein Use-Case-Diagramm für die Beschreibung der Aufgaben des Zelleditors verwendet.

2.3.2 Design Patterns

Design Patterns, auch als Entwurfsmuster bezeichnet, sind wiederverwendbare Lösungsansätze für bestimmte Problemstellungen der objektorientierten Programmierung.

Aus den Erfahrungen der Lösbarkeit von immer wieder, in mehr oder weniger abgewandelter Form, auftretenden Problemestellungen bei objektorientierten Programmieraufgaben können bestimmte nützliche Entwurfsmuster kategorisiert, strukturiert und beschrieben werden. Ein Entwurfsmuster stellt keine Universallösung für bestimmte Probleme dar, sondern beschreibt lediglich einen Lösungsansatz, der sich bei unterschiedlichen Programmieraufgaben bereits als erfolgreich erwiesen hat, und einem Programmierer für die Lösung seiner eigenen Probleme hilfreich sein kann.

Eine der bekanntesten Zusammenstellungen von Design-Patterns ist in [GHJV98] zu finden, die in dieser Arbeit verwendeten Muster sind in diesem Buch beschrieben und werden nun vorgestellt.

2.3.2.1 Das Visitor-Pattern

In [GHJV98] wird als Motivation für den Einsatz des Visitor-Patterns die Möglichkeit genannt, neue Funktionen für Elemente hinzuzufügen, ohne notwendigerweise die Elementklassen ändern zu müssen.

Wenn auf unterschiedliche Objekte einer Datenstruktur bestimmte Operationen ausgeführt werden sollen, deren Behandlungsroutinen von der konkreten Klasse des Objektes abhängen, kann das Visitor-Pattern eingesetzt werden. Anstatt für jede Klasse eine neue Operation zu implementieren, werden die Methoden durch den Besuch eines Visitor-Objektes abgearbeitet. Die Methoden sind von den Klassen getrennt, für veschiedene Operationen werden konkrete Visitor-Klassen implementiert. Die Struktur dieses Patterns wird in Abb. 2.20 dargestellt.

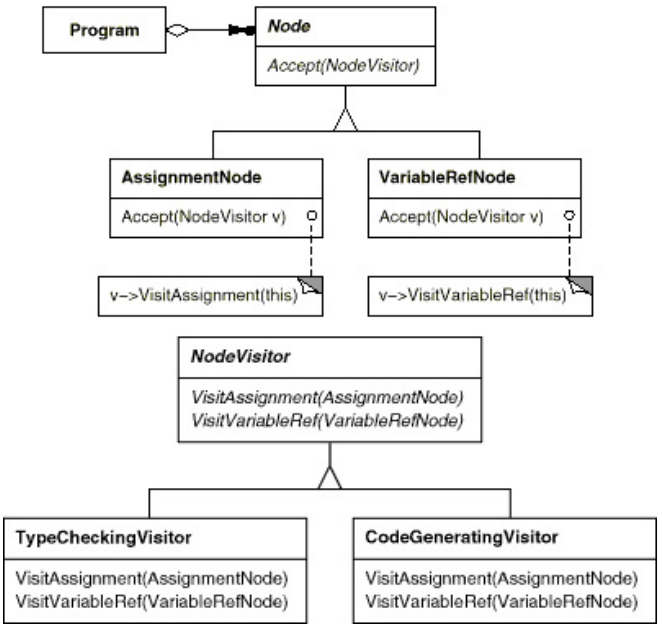

Abbildung 2.20: Klassendiagramm des Visitor-Patterns [GHJV98]

2.3.2.2 Das Prototype-Pattern

Das Prototype-Pattern wird verwendet, um aus einer vorhandenen Instanz einer Klasse eine neue kopierte Instanz zu erstellen [GHJV98]. Jede konkrete Prototype-Klasse implementiert eine Funktion, die dazu verwendet werden kann, einen Klon einer Instanz dieser Klasse herzustellen. Prototypen werden eingesetzt, wenn Objekte dynamisch zur Laufzeit erstellt werden sollen, ohne dass deren konkrete Klasse bekannt sein muss (siehe dazu Abb. 2.21).

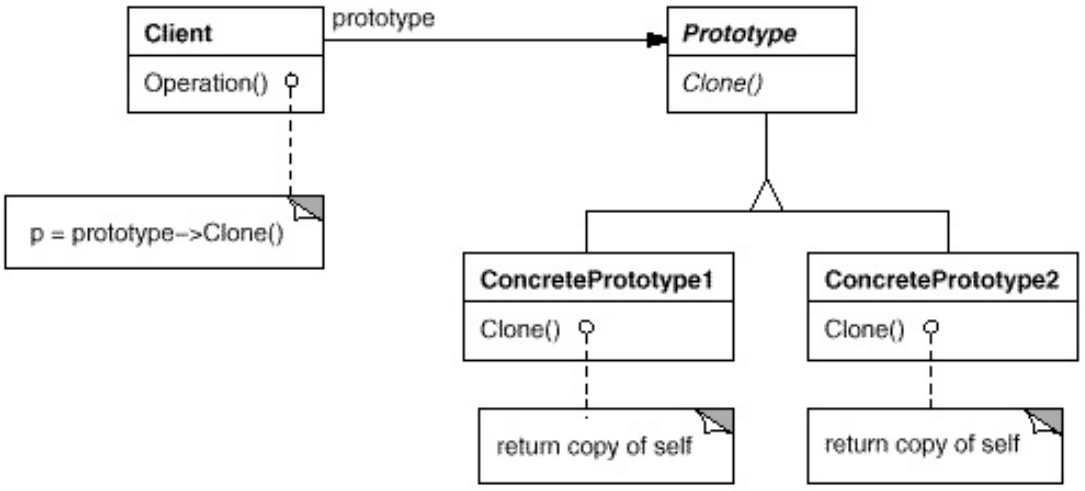

Abbildung 2.21: Klassendiagramm des Prototype-Patterns [GHJV98]

2.3.2.3 Das Observer-Pattern

Dieses Designmuster wird eingesetzt, wenn mehrere Objekte existieren, die alle vom Zustand eines einzigen Objektes abhängen. Häufig können diese Objekte selbst den Zustand der zentralen Instanz ändern, bei einer Änderung dieses Zustandes müssen alle abhängigen Objekte über diese Änderung benachrichtigt werden, um sich entsprechend an die neuen Bedingungen anpassen zu können.

Die Schlüsselobjekte dieses Designmusters sind das Subjekt, das zentrale Element und die vom Zustand des Subjekts abhängigen Observer. Die Struktur des Observer-Patterns wird in Abbildung 2.22 dargestellt.

2.3.2.4 Das Model-View-Controller-Konzept

Das Model-View-Controller-Konzept (kurz: MVC-Konzept) ist ein weitverbreitetes Designmuster, das insbesondere bei der Entwicklung von grafischen Benutzerschnittstellen verwendet wird.

Dieses Konzept wird eingesetzt, um die Daten einer Anwendung von der Präsentation der Daten logisch zu trennen [Ulle02]. Das „Model“ speichert die Daten einer Anwendung und bietet Methoden, um die Daten auslesen oder manipulieren zu können. Die „Views“ lesen über die durch das „Model“ bereitgestellten Methoden die Daten aus, bereiten sie auf und stellen sie in bestimmter Art und Weise dar. Der

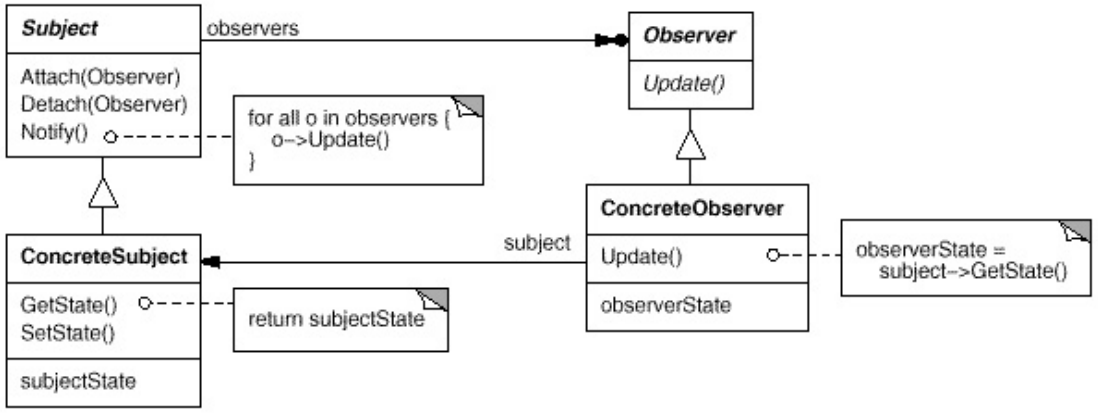

Abbildung 2.22: Klassendiagramm des Observer-Patterns [GHJV98]

„Controller“ verwaltet die verschiedenen „Views“ und kümmert sich um die Abarbeitung von Benutzereingaben, durch die die Daten des „Models“ verändert werden können.

Um einen Mechanismus zu implementieren, durch den es möglich ist, die einzelnen „Views“ bei Änderung des „Models“ direkt zu benachrichtigen, wird das MVC-Konzept häufig mit dem im vorigen Abschnitt beschriebenen Observer-Pattern kombiniert. Das Subjekt des Observer-Patterns entspricht dabei dem „Model“ des MVC-Konzeptes, die unterschiedlichen „Views“ sind die Observer.

3 Stand der Technik

3.1 Simulation von Zellen

Die mathematische Modellierung und Simulation von Zellprozessen ist ein Aufgabenbereich des Forschungszweiges Systembiologie. Die Systembiologie beschäftigt sich mit der Analyse der Wechselwirkungen aller Elemente lebender Syteme [IdGH01]. In [Kita00] werden vier zu untersuchende Bereiche der Systeme definiert. Ein Bereich behandelt die Systemstrukturen, die Bestandteile der Systeme und ihre strukturellen Zusammenhänge, ein weiterer die Systemdynamik, die Veränderung des Systems im zeitlichen Verlauf. Von Interesse sind außerdem die Systemkontrollen, d. h. alle Mechanismen die kontrollierend auf das biologische System einwirken und schließlich die Konstruktionsprinzipien des Systems.

In [IdGH01] ist eine Vorgehensweise bei der Analyse des Systems der Zelle aufgezeigt. Bei dem beschriebenen Ansatz wird zu Anfang das System systematisch gestört, die Veränderungen auf Protein-, Genom- und Signalebene aufgezeichnet und in einem Datenbestand gesammelt. Aus diesen Daten wird dann ein mathematisches Modell entwickelt, das eine abstrakte Darstellung der Struktur des Systems beschreibt und Vorhersagen über Veränderungen bei Störungen des Systems ermöglicht.

Die Aufgabenbereiche Systembiologie erfordern Anstrengungen in unterschiedlichen Forschungszweigen, wie der Bio- und Ingenieurwissenschaften, der Mathematik, Informatik und der Systemtheorie.

Ein große Ziel der Systembiologie, die vollständige Modellierung einer Zelle und das Verständnis aller Zellprozesse und dadurch die Bereitstellung einer Möglichkeit zum Formulieren von Vorhersagen von Reaktionen der Zelle auf beliebige Störungen, ist noch eine Zukunftsvision. Es gibt allerdings schon einige Ansätze zum Verständnis von einzelnen Teilprozessen der Zelle und auch zur Entwicklung von Computermodellen ganzer Zellen.

Viele laufende Projekte und Arbeiten beschränken sich auf die Untersuchung von Teilbereichen der Zelle oder einzelnen Zellprozessen. So beschäftigt sich [HeSS93] z. B. mit dem Verhalten von Lipid-Molekülen in Membranen, [OnIk00] behandelt

die Teilprozesse der Selbsterhaltung und Selbstreproduktion von Zellen. Allerdings gibt es auch Ansätze, die versuchen die Zelle als ganzes System zu erfassen.

Ein Forschungsprojekt, das sich mit der Entwicklung eines umfassenden Zellmodells befasst, ist das E-Cell-Projekt [ECel]. Dieses Projekt wurde 1995 gestartet, mit dem erklärten Ziel der Erschaffung eines in silicio-Modells einer kompletten Zelle. Das erste Zellmodell war noch eine hypothetische Zelle, im Laufe der Zeit wurde dann die Modellierung einfacher echter Zellen als Ziel erklärt. In dem E-Cell-Projekt wird eine objektorientierte Softwareumgebung zum Modellieren, Simulieren und Analysieren komplexer Zellsysteme entwickelt, das E-Cell-System (siehe Abb. 3.1).

Ein weiteres Forschungsprojekt, das sich der Modellierung ganzer Zellen widmet, ist das Virtual Cell Project [VCel]. Auch in diesem Projekt wird eine Softwareumgebung für die Zellsimulation entwickelt (siehe Abb. 3.2).

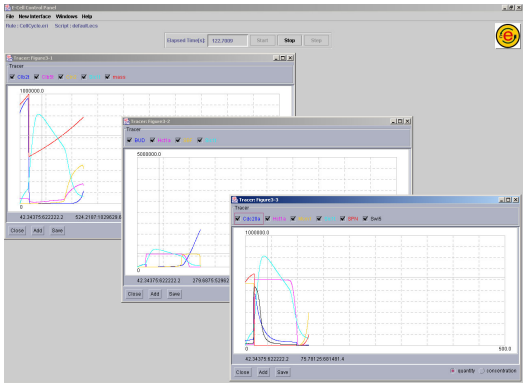

Abbildung 3.1: Das Programm E-Cell2 [ECel]

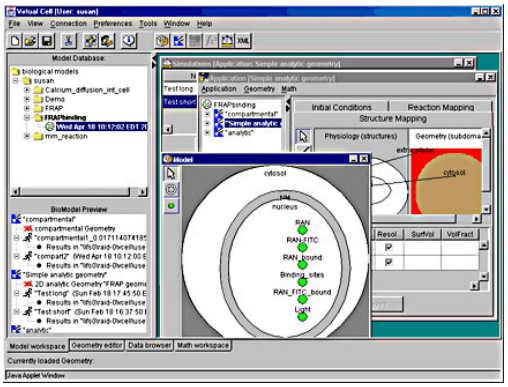

Abbildung 3.2: Das Programm Virtual-Cell [VCel]

Dreidimensionale Darstellungen zur Visualisierung der Zelle und von simulierten Zellprozessen spielen bei vielen Projekten noch keine große Rolle. Dies liegt zumeist daran, dass die Schwerpunkte der Untersuchung bei der Analyse und Darstellung von biochemischen Prozessen und des Reaktionsnetzwerkes liegen und räumliche Bewegungsprozesse aufgrund fehlender Methoden und Werkzeuge im Bereich der Biologie und der Datenerfassung, aber auch im Bereich der computerbasierten Modellierung und Simulation, noch keine entscheidende Rolle spielen. Zur Darstellung der Ergebnisse und des Simulationsverlaufs genügen daher häufig Graphen, Diagramme oder abstrakte 2D-Modelle.

Ein Beispiel für die Verwendung von 3D-Modellen bei der Erforschung von Zellprozessen wird in [MCDP+96] dargestellt. Durch die dreidimensionalen Visualisierung von Proteinen in zellulären Interaktionsprozessen soll es ermöglicht werden, wechselseitige Interaktionen von Proteinen zu bestimmten Zeitpunkten in Relation zu ihrer räumlichen Lage bestimmen zu können.

3D-Darstellungen werden allerdings oftmals nur für statische Modelle oder zur Visualisierung von Ergebnisdaten verwendet. Sie dienen dabei als Vorzeige- oder Lehrobjekte, die animiert oder betrachtet, aber in keine dynamischen Simulationsprozesse eingebunden werden können. Es handelt sich bei diesen Modellen um reine Grafikobjekte, die keine Informationen über die dargestellten Objekte besitzen, weswegen sie für Simulationsprozesse nicht zu verwenden sind (siehe Abb. 3.3 und Abb. 3.4).

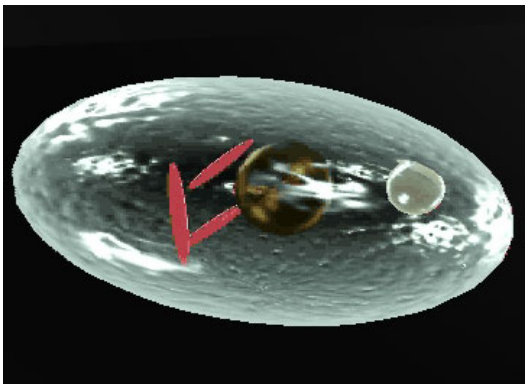

Abbildung 3.3: Animiertes 3D-Modell einer Zelle [ICel]

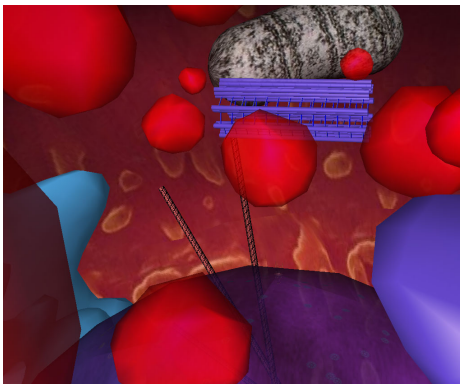

Abbildung 3.4: VRML-Modell einer Zelle [Bio]

3.2 Computerbasierte 3D-Modellierung

Es gibt eine ganze Reihe von umfangreichen Modellierungswerkzeugen, die zur Erzeugung von computergraphischen 3D-Objekten verwendet werden können. Mit Programmen wie 3D-Studio Max oder Maya können umfangreiche virtuelle Welten erstellt werden. In [Bio] steht ein unter 3D-StudioMax generiertes Modell einer Zelle zur Verfügung.

Diese Modellierungswerkzeuge sind allerdings reine 3D-Modellierer und können nicht eingesetzt werden, um Modelle zu erstellen, die neben der 3D-Grafik auch das Wissen über das Objekt, das sie darstellen, besitzen.

3.3 Szenengraph-APIs

Als Szenengraph-API werden Programmierbibliotheken bezeichnet, durch die grafische 3D-Anwendungen entwickelt werden können, die zur Verwaltung der 3D-Grafiken Szenengraphen verwenden. Szenengraph-APIs für C++ in Verbindung mit OpenGL sind z. B. OpenGL Performer, OpenInventor, OpenSG oder OpenSceneGraph. Für die Entwicklung des Zelleditors standen die beiden letztgenannten APIs zur Auswahl. Es wurde die OpenSceneGraph-API verwendet, da diese Bibliothek auch bei anderen nahestehenden Projekten in Gebrauch ist.

Das OpenSceneGraph-Projekt [Osfib] wurde 1998 von Don Burns gestartet und September 1999 in der Projektleitung durch Robert Osfield übernommen. OpenSceneGraph ist eine plattformübergreifende OpenSource (LGPL) Szenengraph-API, die auf Standard-C++ und OpenGL basiert.

OpenSceneGraph besitzt eine große Community und eine aktive Mailing-Liste, wird allerdings ansonsten nur spärlich durch Dokumentationen unterstützt. Haupteinsatzgebiete der OpenSceneGraph-API sind Flugsimulatoren, Spiele, Virtual Reality-Anwendungen und Visualisierungen wissenschaftlicher Daten.

3.4 Windowing Toolkits

Ein Windowing Toolkit ist in der objektorientierten Programmierung eine Klassenbibliothek, durch die die Entwicklung von grafischen Benutzerschnittstellen verein-

facht wird. Für die Programmiersprache C++ existieren mehrere Windowing Toolkits, häufig verwendet werden die Microsoft Foundation Classes (MFC), die Qt-Bibliothek, wxWindow oder der FOX Windowing Toolkit. Bei der Entwicklung des Zelleditors wurde der FOX Windowing Toolkit verwendet, der nun vorgestellt wird.

Der FOX Windowing Toolkit [vdZi] ist ein auf der Programmiersprache C++ basierender, plattformunabhängiger Toolkit zur Entwicklung von grafischen Benutzeroberflächen.

Mit der Entwicklung des FOX Windowing Toolkit wurde Mitte 1997 von Jeroen van der Zijp begonnen. FOX wurde zu Anfang als OpenSource-Projekt unter Linux entwickelt, allmählich dann auf weitere Unix-Derivate portiert und läuft mittlerweile auch auf den unterschiedlichen Microsoft Windows Plattformen.

Für die Verwendung des FOX Windowing Toolkits zur Entwicklung der Benutzeroberfläche des Zelleditors sprachen mehrere Sachverhalte:

- FOX ist OpenSource und deswegen ohne Lizenzierungskosten verwendbar.
- Der FOX Windowing Toolkit ist plattformunabhängig. Durch die Nutzung des FOX Windowing Toolkits besteht die Möglichkeit, das Programm neben der MS-Windowsumgebung auch für Unix-System zu kompilieren.
- Der FOX Windowing Toolkit bietet Unterstützung für OpenGL und ließ sich dadurch relativ einfach für das Rendern der OpenSceneGraph-Daten verwenden.
- Unter FOX lassen sich aus den vorhandenen GUI-Elemente, wie z. B. Fenstern, Buttons, usw., einfach neue Elemente weiterentwickeln, die bei der Gestaltung der Benutzeroberfläche verwendet werden können.

Bei der Entwicklung der Benutzeroberfläche hat FOX gegenüber kommerziellen Windowing Toolkits wie MFC oder Qt allerdings auch einige Nachteile aufzuweisen:

- Im Vergleich zu kommerziellen Produkten kann FOX in Sachen Dokumentation nicht mithalten. FOX besitzt eine informative Internetpräsenz und eine aktive Mailing-Liste, es sind allerdings keine Bücher oder Referenzen über FOX verfügbar, sodass häufig nur der Blick in den Sourcecode bei der Lösung von Problemen weiterhilft.
- Zum Erstellen von FOX-Benutzeroberflächen existieren keine besonderen Entwicklungswerkzeuge, die eine Entwicklung des GUIs per Drag-and-Drop ermöglichen würden, es bleibt nur die mühsame Programmierung per Hand.
- Der FOX-Windowing Toolkit ist stetig in der Entwicklung begriffen, weswegen der Code des Projektes noch einige Fehler enthalten kann und der Funktionsumfang noch nicht vollständig ist.

4 Konzept und Implementierung der Klassen für die computerbasierte Repräsentation einer Zelle

Um eine biologische Zelle im Computer zu erfassen, muss ein Klassenmodell zur Verfügung stehen, durch das die Zelle in einem repräsentativen Datenmodell abgebildet werden kann.

In diesem Kapitel werden die für diesen Zweck entworfenen Klassen vorgestellt und ihre Verwendung beschrieben. Zuerst werden die dem entwickelten Klassenmodell zugrunde liegenden konzeptionellen Überlegungen erläutert, damit die Struktur des entwickelten Klassenmodells besser nachvollzogen werden kann.

Am Ende des Kapitels wird auf die Serialisierung und auf die Erweiterbarkeit des Zellmodells eingegangen.

4.1 Konzeptionelle Überlegungen

Es werden nun die Überlegungen aufgeführt, auf deren Basis die Klassen für das Zellmodell entwickelt wurden Zuerst wird aber der Begriff der Zellmodellierung erläutert.

4.1.1 Zellmodellierung

Der Prozess der Entwicklung einer dreidimensionalen computergrafischen Repräsentation eines beliebigen physischen Objektes wird gemeinhin als Modellierung bezeichnet.

Der in diesem Dokument verwendete Begriff der Zellmodellierung beschreibt den Vorgang der Entwicklung eines 3D-Computermodells einer Zelle. Er bezieht sich allerdings nicht ausschließlich auf die Entwicklung der 3D-Repräsentation der Zelle,

sondern umfasst auch die semantische Modellierung der Zelle; im Gegensatz zu einem rein grafischen 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.

4.1.2 Bausteine

Als Bausteine werden alle Elementarobjekte bezeichnet, die zur Modellierung der Zelle eingesetzt werden können. Jeder Baustein besitzt eine 3D-grafische Repräsentation im Modell und kann dort positioniert werden. Bausteinen kann ein konkreter Typ zugeordnet werden, z. B. der Typ Zellkern oder der Typ Mitochondrium, der durch bestimmte Attribute, ein spezifisches Verhalten und einen räumlichen Aufbau beschrieben wird.

Das komplette Modell der Zelle wird aus den Bausteinen zusammengesetzt. Welche Bausteine für diese Aufgabe zur Verfügung stehen, hängt von den durch den Verwendungszweck der modellierten Zelle gegebenen Erfordernissen ab. Grundsätzlich kann jede der in Abschnitt 2.1.3.1 beschriebenen 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 Unterbausteine — ein Zellkern kann aus zwei Membran-Bausteinen und einem Nucleolus-Baustein zusammengesetzt werden — unterteilt sich die Zelle in mehrere Modellierungsebenen (siehe Abb. 4.1, 4.2). In der untersten Ebene befindet sich das Modell der Zelle selbst, mit den beschriebenen Zellkomponenten als möglichen Bausteinen. Die nächsthöhere Ebenen 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 Abb. 4.3). 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 Blätter 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.

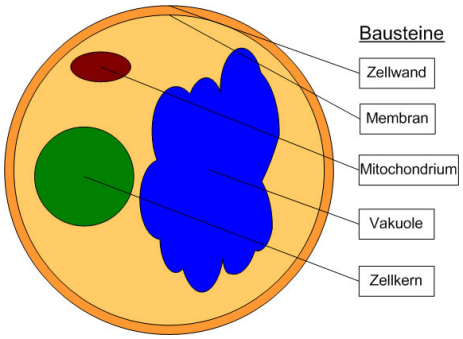

Abbildung 4.1: Bausteine der Modellierungsebene Zelle

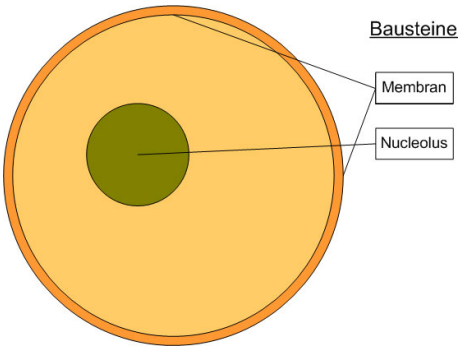

Abbildung 4.2: Bausteine der Modellierungsebene Zellkern

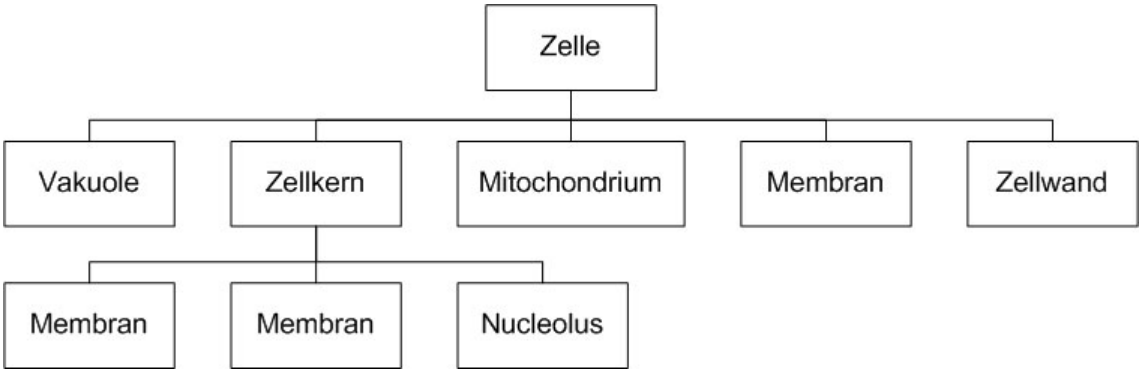

Abbildung 4.3: Modellierungsebenen der Zelle, dargestellt in einem Baumdiagramm

4.1.2.1 Klassifizieren von Zellstrukturen

Es wird nun versucht die Bausteine 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 Tabelle 4.4 nochmal zusammengestellt.

Strukturtyp	Beispiel
begrenzt	Zellkern, Mitochondrien, usw.
begrenzend	Membran, Zellwand
unbegrenzt	Dictyosom, Proteinfilamente, Ribosomen
komplex	Zytoskelett, Golgi-Apparat

Abbildung 4.4: Auflistung der verschiedenen Strukturtypen

4.1.2.2 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 werden soll, 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 3D-Modell transformiert werden können, relativ zu anderen Bausteinen bewegt oder rotiert werden, dann muss es ebenfalls durch einen Baustein repräsentiert werden.

4.1.3 3D-Ansicht der Bausteine

Jeder der im vorigen Kapitel vorgestellten Bausteine besitzt eine dreidimensionale Ansicht im Zellmodell. Das 3D-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.

4.1.3.1 Darstellungsmethoden

Um dreidimensionale Objekte im Computer darzustellen, existieren unterschiedliche Modellierungsmethoden. Die 3D-Repräsentation der Zelle und anderer aus mehreren Unterbausteinen zusammengesetzten Bausteine wird aus den grafischen Darstellungen der Untereinheiten gebildet. Diese Art der Modellierung entspricht der in Abschnitt 2.2.1.1 beschriebenen 3D-Repräsentationsmethode durch Elementarobjekte, wobei in diesem Fall die Bausteine die Elementareinheiten darstellen.

Damit für die zusammengesetzten Baustein-Modelle eine 3D-Darstellung existiert, müssen mindestens die Bausteine, die keine Unterbausteine besitzen, also alle Bausteine, die die Blätter des Modellierungsbaums bilden, ein eigenes Aussehen besitzen.

Um dieses Aussehen zu definieren, wird der Repräsentationsansatz durch Elementarobjekte mit der Flächenmodell-Methode kombiniert: 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 grafische Darstellung einzelner Bausteine (siehe auch Abb. 4.5):

- 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 Flächenmodell eines Elementarobjektes 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, 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.

4.1.3.2 Elementarobjekte

Ein Elementarobjekt ist eine durch verschiedene Parameter eindeutig definierte Form, die bei der Entwicklung der 3D-Ansichten der Bausteine eingesetzt werden kann.

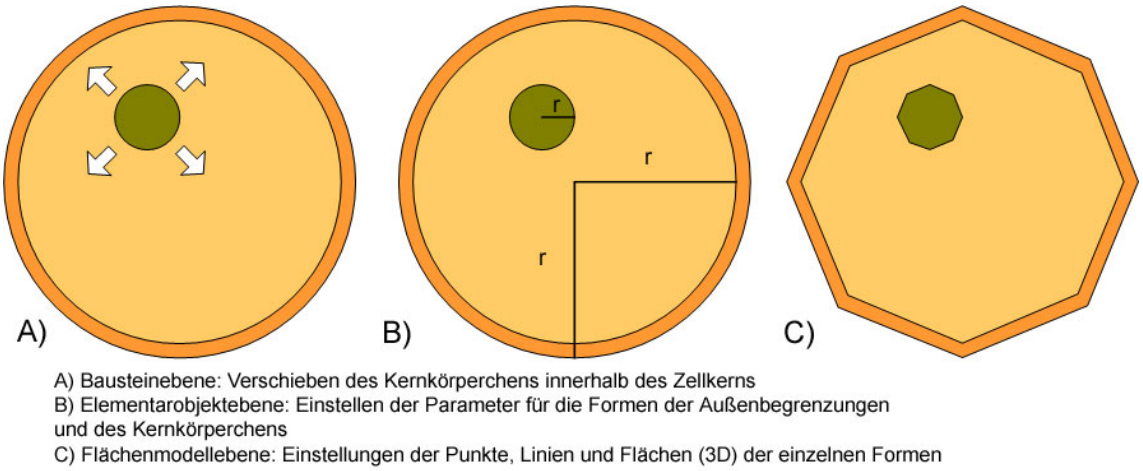

Abbildung 4.5: Darstellungsebenen des Zellkerns

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 zumindestens Skalierungsparameter für die drei Raumachsen spezifiziert werden. In Abb. 4.6 sind verschiedene Formen und die dazugehörigen Parameter dargestellt.

Zur Beschreibung der Geometrie des Elementarobjekte wird ein Flächenmodell genutzt. Damit dieses Flächenmodell für alle Objekte einheitlich aufgebaut ist, gelten für die Beschreibung folgende Richtlinien:

- Das Flächenmodell besteht ausschließlich aus Dreiecksflächen. Der Vorteil der Triangularisierung eines Flächenmodells ist dadurch begründet, dass die drei Eckpunkte eines Dreiecks immer in einer Ebene liegen und dadurch beliebig im Raum verschoben werden können.
- Alle Punkte des Modells werden gemeinsam gespeichert.
- Die einzelnen Punkte können in Dreiergruppen indiziert werden, wodurch beschrieben wird, welche Punkte die Dreiecke des Flächenmodells bilden.
- Für jeden Punkt des Modells wird ein Normalvektor gespeichert. Der Normalvektor eines Punktes wird als arithmetisches Mittel der Normalvektoren aller Flächen, die den Punkt als Eckkoordinate besitzen, berechnet.
- Damit eine Textur über das Objekt gelegt werden kann, muss eine entsprechende Textur-Kachelung der Flächen des Modells spezifiziert werden.

4.1.3.3 Der Szenengraph

Die grafische 3D-Darstellung des Modells wird in einem Szenengraph gehalten. Jeder Baustein des Modells besitzt eine eigene 3D-Repräsentation, die in einem Teilbereich des Graphen untergebracht ist. Der Aufbau des Teilgraphen der Bausteine und die

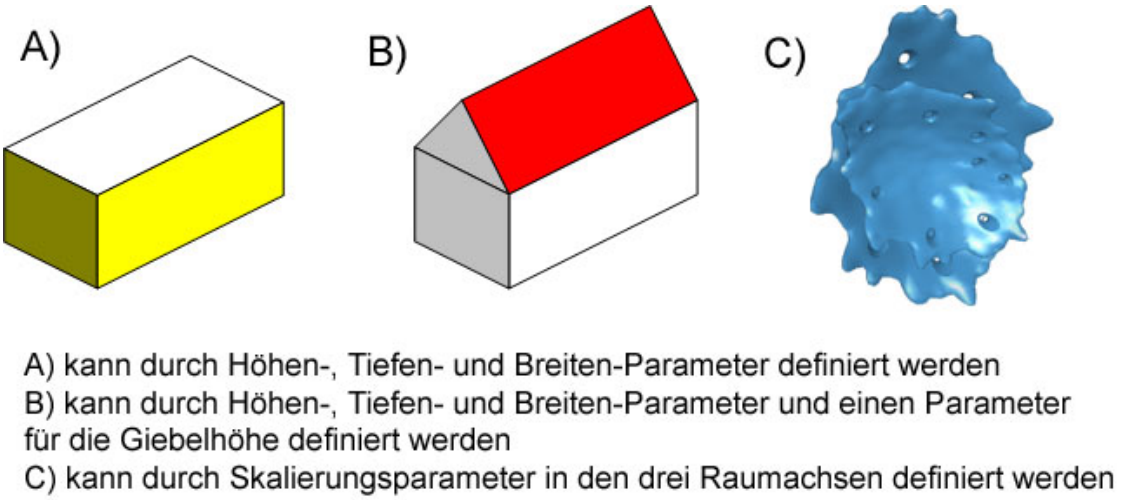

Abbildung 4.6: Parametrisierung von Elementarobjekten

Zusammensetzung des gesamten Szenengraphen wird 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 4.7 wird der Aufbau des Szenengraphen des in Abb. 4.3 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 3D-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 bausteineigene 3D-Ansicht soll, wie in Abschnitt 4.1.3 beschrieben, durch Elementarobjekte modelliert werden.

Die geometrischen Informationen dieser Elementarobjekte werden in einem Geometrieknoten untergebracht, der sich wiederum in einem Transformationsknoten befindet. Um ein Elementarobjekt in die Baustein-Ansicht 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 wer-

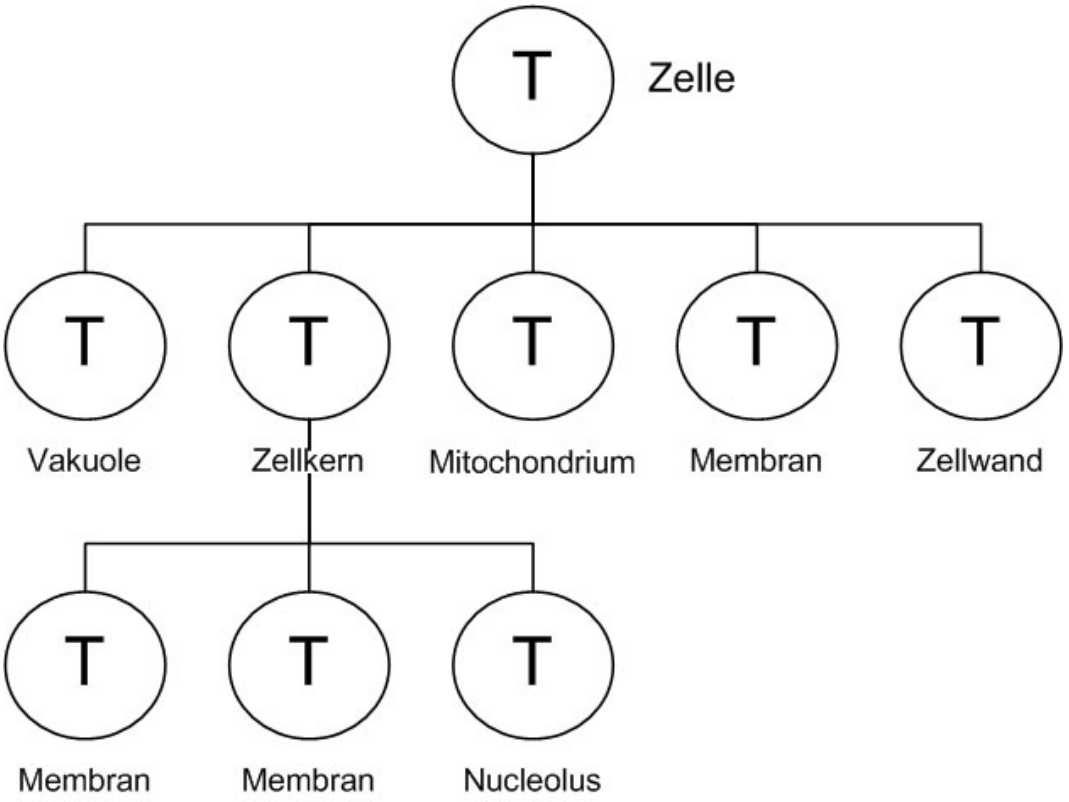

Abbildung 4.7: Aufbau des Szenengraphen des Modells aus Abb 4.3; *T - Transformationsknoten*

den. Der Aufbau des Szenen-Teilgraphen eines Bausteins ist in Abb. 4.8 aufgeführt.

Durch den bisherigen Aufbau des Szenengraphen werden lediglich die Geometrien der Bausteine beschrieben. Der Aufbau wird im nächsten Abschnitt um Material- und Darstellungseigenschaften erweitert.

4.1.3.4 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 alle unterschiedliche Eigenschaften besitzen, in manchen Fällen ist es allerdings erwünscht für alle Elementarobjekte eines Bausteins, oder sogar für die gesamten Ansichten aller Unterbausteine diesselben Material- und Darstellungseinstellungen anzuwenden.

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 Abb. 4.9):

- 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

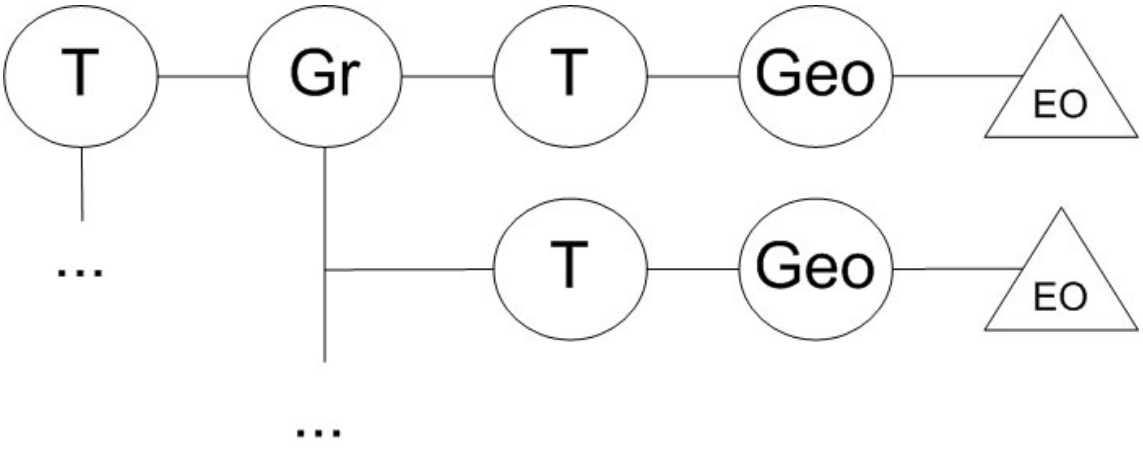

Abbildung 4.8: Aufbau des Szenengraphen eines Bausteins; *T - Transformationsknoten, Gr - Gruppenknoten, Geo - Geometrieknoten, EO - Elementarobjekt*

also die komplette 3D-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.

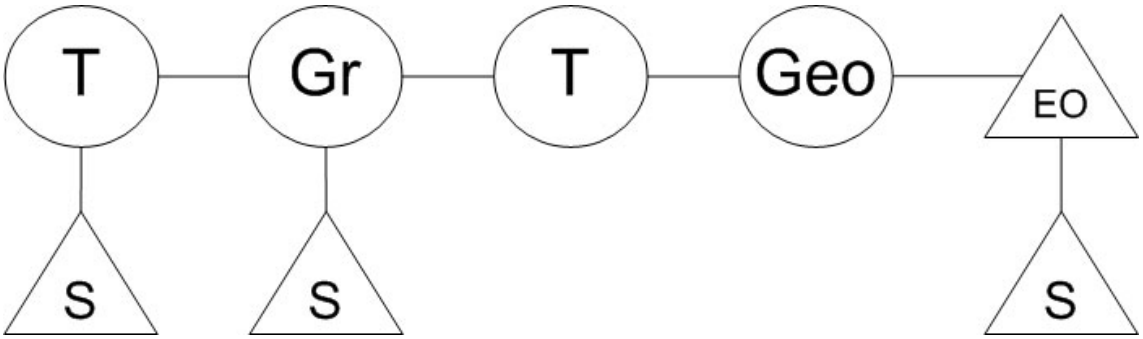

Abbildung 4.9: Aufbau des Szenengraphen eines Bausteins mit Material-und Darstellungseigenschaften; *T - Transformationsknoten, Gr - Gruppenknoten, Geo - Geometrieknoten, EO - Elementarobjekt, S - Eigenschaftszustand*

4.1.4 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, z.B. über Form oder Größe oder Volumen eines Bausteins aufgestellt werden.

Prinzipiell ist der Benutzer für die Korrektheit der modellierten Objekte verantwortlich. Modellierungsregeln, die sich abstrakt formulieren lassen, können allerdings in die Programmlogik eingebunden werden, sodass 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, ihn auf der anderen Seite aber noch genug Freiheiten zur individuellen Gestaltung zu bieten.

4.1.5 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. Die Abbildungen 2.2 und 2.3 stellen abstrakte Modelle einer Pflanzen- und einer Tierzelle dar und verdeutlichen den unterschiedlichen Aufbau dieser zwei Zelltypen.

Eine Konsequenz dieser Typisierung der Zelle ist, dass die im vorherigen Abschnitt beschriebenen Modellierungsrichtlinien zelltypabhängig formuliert werden müssen.

Die Klassifizierung der Zellen wird in Kapitel 2.1.2 beschrieben. In Abbildung 4.10 ist ein Auschnitt 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.

4.2 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 3D-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 grafischen Eigenschaften und die Schnittstellen zur Manipulation der Grafik der Bausteine werden in einer speziellen Klasse *View* gekapselt. Durch die Klasse *Shape*

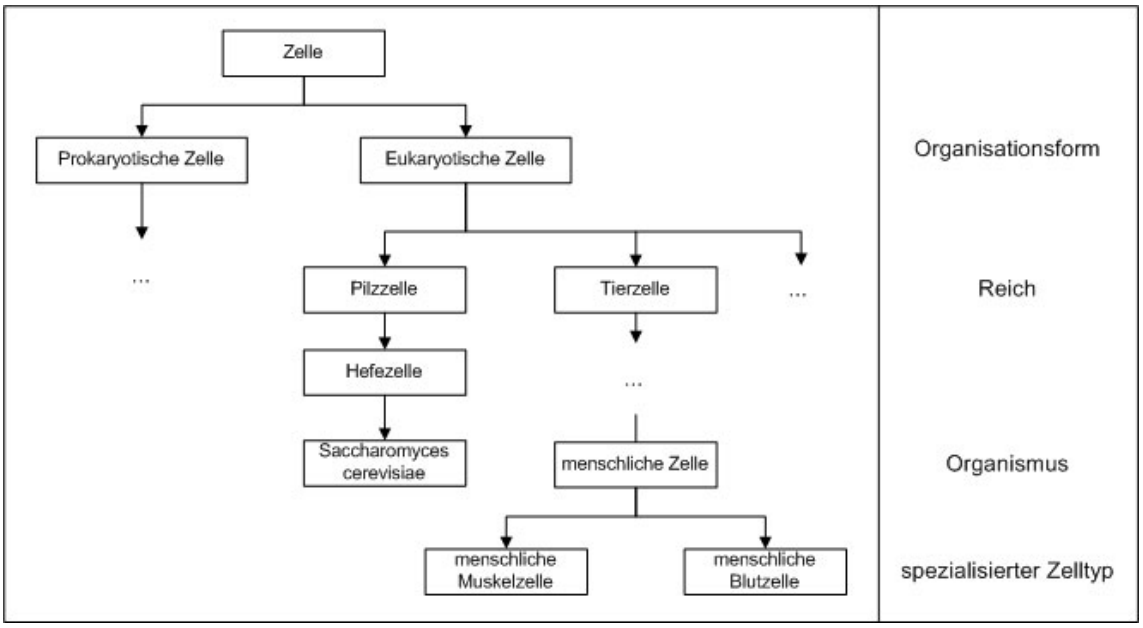

Abbildung 4.10: Ausschnitt eines Klassifizierungbaumes von Zellen

können bei der Baustein-Modellierung eingesetzte Elementarobjekte, parametrisierbare Formen oder Körper, beschrieben werden.

Zur Einführung der in Abschnitt 4.1.4 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, klassifizierungabhängige Richtlinien für beliebige Bausteine aufstellen zu können.

Der Aufbau der soeben vorgestellten Klassen wird in den folgenden Abschnitten genauer erläutert.

4.2.1 Bausteinklassen

In diesem Abschnitt werden die Klassen vorgestellt, die zur Repräsentation der in Abschnitt 4.1.2 beschriebenen Bausteine entworfen wurden.

4.2.1.1 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 in Abschnitt 4.1.2.1 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 grafischen Eigenschaften verwaltet werden.

- Jedes Baustein-Objekt besitzt ein Objekt der Klasse ComponentManager, das die Modellierungsrichtlinien definiert.

- Für den in Abschnitt 4.3.1 beschriebenen Klon-Mechanismus wird in der Component-Klasse 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, wie in Abschnitt 4.3.2 beschrieben wird.

- Um den Abschnitt 4.2.4.1 entwickelten Visitor-Mechanismus anwenden zu können, wird eine virtuelle accept-Funktion bereitgestellt.

4.2.1.2 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 3D-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.

4.2.2 Klassen zum Beschreiben des grafischen Auftretens

Das in Kapitel 4.1.3 beschriebene Konzept für die 3D-Darstellung der Zellbausteine wird durch die Klassen Shape und View realisiert. Die Zusammenhänge zwischen den beiden im Folgenden beschriebenen Klassen zur gafischen Darstellung und den im vorigen Abschnitt beschriebenen Baustein-Klassen, wird in Abb. 4.11 in Form eines Klassendiagramms dargestellt.

4.2.2.1 Die Klasse Shape

Ein Shape-Objekt beschreibt eine durch Parameter definierbare geometrische Form. Shape-Objekte werden als Elementareinheiten bei der Modellierung der 3D-Darstellung der Bausteine verwendet.

Die Klasse Shape ist eine Spezialisierung der OSG-Geometry-Klasse und kann dadurch direkt in einen OSG-Szenegraphen eingefügt werden. Bei der Darstellung der Geometrie wurde auf die in Abschnitt 4.1.3.2 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 hängt 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 3D-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 neuimplementieren. Die init-Funktion wird bei der Neuinitialisierung der 3D-Geometrie eines Shape-Objektes und bei der Änderung von Parametern aufgerufen.

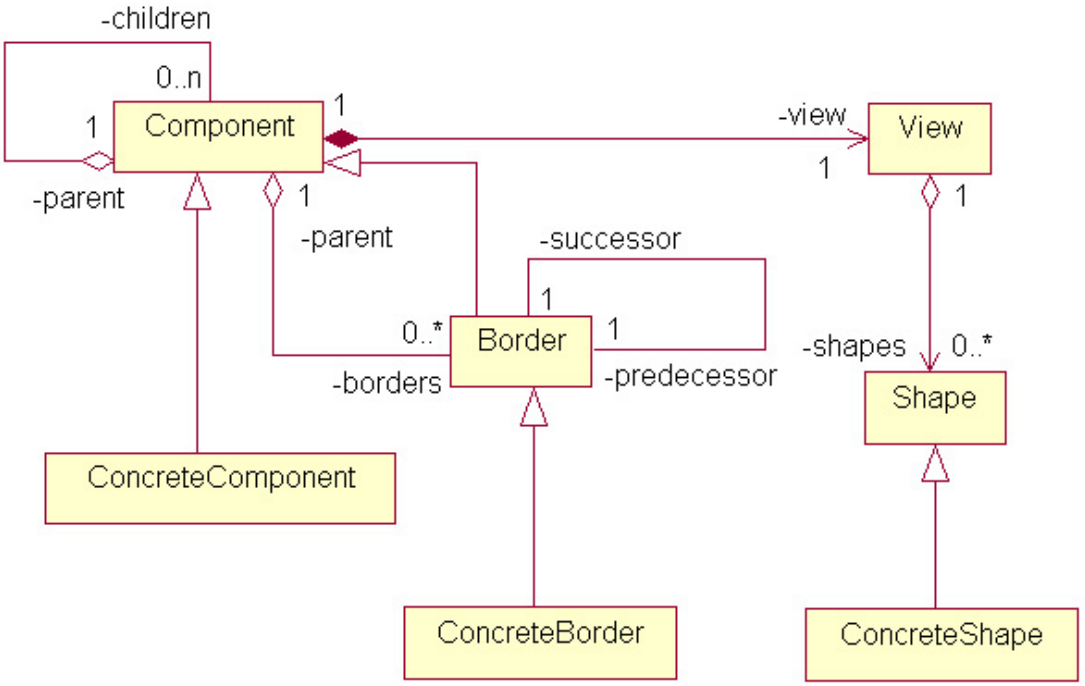

Abbildung 4.11: Klassendiagramm zum Verdeutlichen der Zusammenhänge zwischen Baustein-, View- und Shape-Klassen

4.2.2.2 Die Klasse View

Die Klasse View verwaltet das grafische 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 3D-Darstellung enthält
- und auf alle Shape-Objekte, die die 3D-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 3D-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

4.2.3 Managerklassen

Es werden in diesem Abschnitt die Klassen vorgestellt, die zur Beschreibung der in Abschnitt 4.1.4 erläuterten Modellierungsrichtlinien entwickelt wurden.

4.2.3.1 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 Bausteine und wieviele 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 4.12 werden die Modellierungsrichtlinien für einen normalen Zellkern aufgelistet.

ComponentManager - Zellkern

ComponentRestrictions

Name	Min	Max	Ø
Nucleolus	1	1	1

BorderRestrictions

Name	Index
Membrane	1
Membrane	2

Abbildung 4.12: ComponentManager eines Zellkerns mit einem Kernkörperchen und einer Doppelmembran

BorderManager – innere Membran des Zellkerns

ShapeRestrictions

Name	ParameterRestrictions			
Sphere	Name	Min	Max	Ø
	Radius	0.5	500	5

BorderManager – äußere Membran des Zellkerns

InheritViewFromPredecessor	true

DistanceToPredecessorRestriction

Name	Min	Max	Ø
Distance	0.01	10	0.5

Abbildung 4.13: 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.
- Der augenblickliche Wert des Parameters.

Bei der Spezifikation der Parameterwerte muss auf Beibehaltung der Verhältnissmäßigkeiten 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.

4.2.3.2 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 grafischen 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 4.13 werden die Modellierungsrichtlinien für die beiden Membranen eines Zellkerns aufgelistet. Die Zusammenhänge zwischen den einzelnen Manager- und den Restriction-Klassen werden in Abb. 4.14 dargestellt.

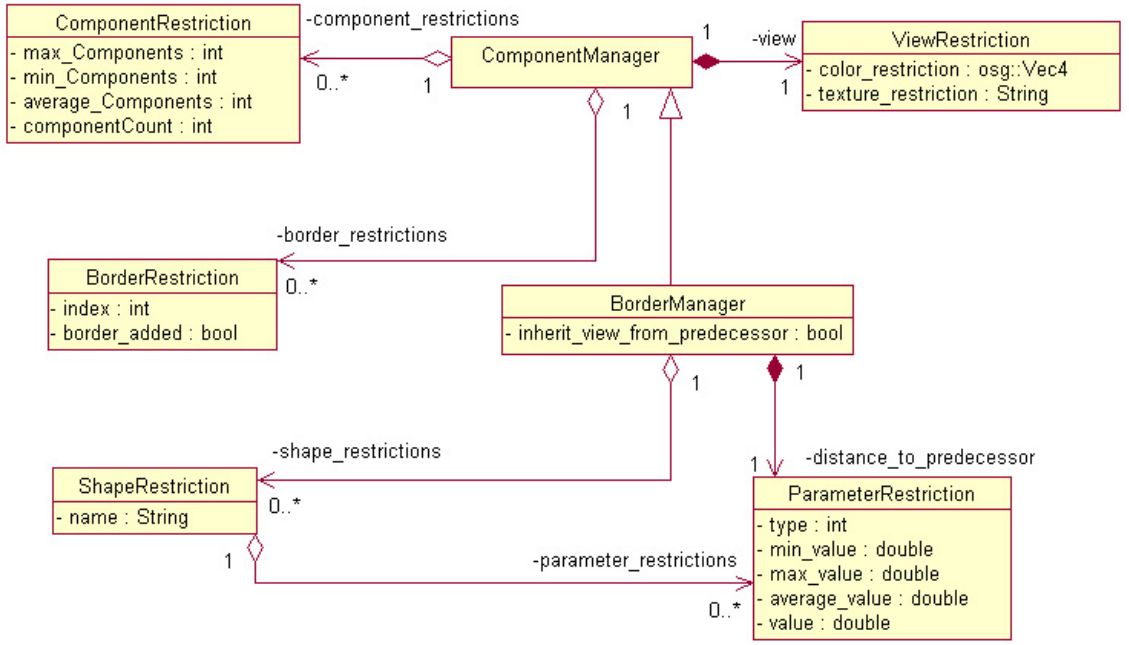

Abbildung 4.14: Klassendiagramm der Managerklassen

4.2.4 Klassen zur Auflösung des Problems der unterschiedlichen Zelltypen

Damit der in Kapitel 4.1.5 dargestellte unterschiedliche Aufbau von Zellen unterschiedlichen Typs bei der Modellierung berücksichtigt werden kann, wurden einige Klassen entwickelt, die nun kurz vorgestellt werden.

4.2.4.1 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 ComponentManage-Einstellungen eigenständig verwaltet. Ein bedeutender Nachteil dieser Methode ist allerdings, dass für die Bereitstellung neuer Zelltypen alle Bausteinklassen angepasst werden müssten (siehe Abb. 4.15).

Ein weiterer Ansatz besteht darin, für jeden Baustein typabhängige Klassen bereitzustellen, z.B. eine Klasse YeastNucleus um einen Zellkern einer Hefezelle, oder eine Klasse HumanMuscleMitochondrion um eine Mitochondrium einer menschlichen Muskelzelle darzustellen. Auch bei diesem Ansatz erfordert die Erweiterung modellierbarer Zelltypen einen erheblichen Implementierungsaufwand (siehe Abb. 4.16).

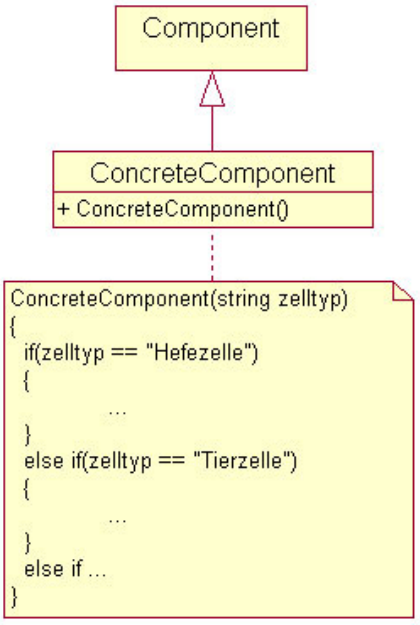

Abbildung 4.15:

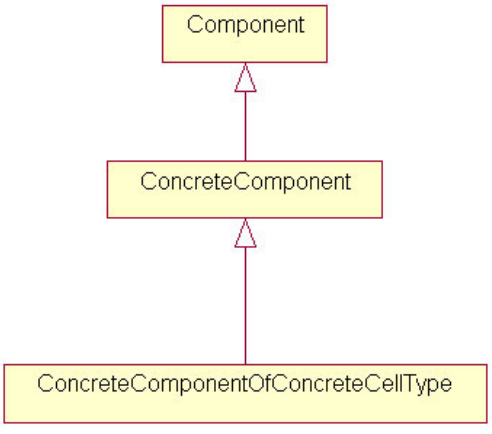

Abbildung 4.16:

Für das entwickelte Klassenmodell wurde eine dritte Methode, die auf dem im Kapitel 2.3.2.1 beschriebenen Visitor-Pattern basiert, entwickelt. 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. Die innere Membran eines Mitochondriums z. B. unterscheidet sich 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 4.17 wird der entwickelte Mechanismus anhand eines Klassendiagramms dargestellt.

4.2.4.2 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 — bei der Zelle sind dies z. B. die Klassifizierung nach Organismus auf der einen und nach spezialisiertem Zelltyp auf der anderen Seite — in einem einzigen Klassifizierungsbaum unterzubringen (siehe Abb. 4.18)

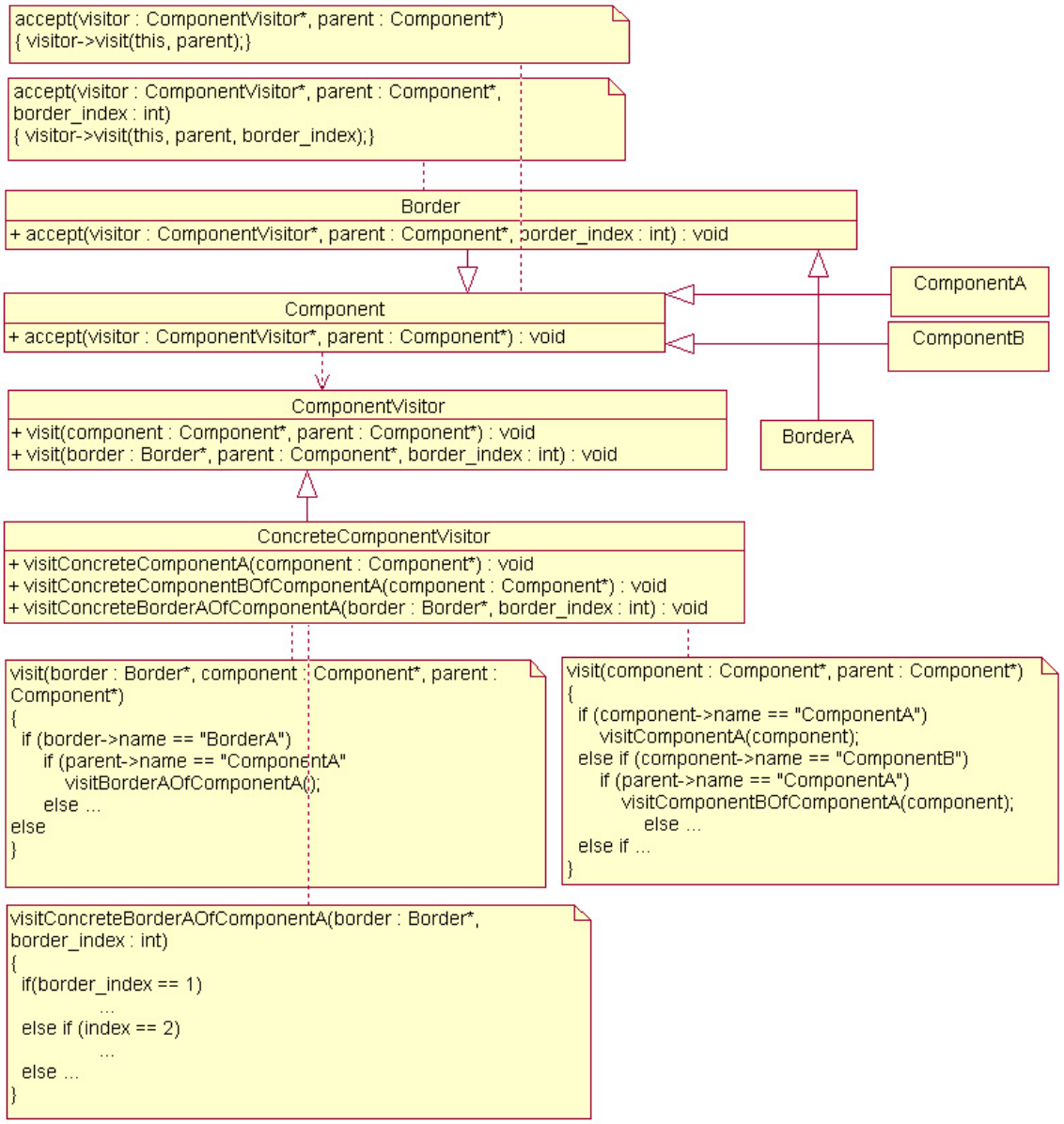

Abbildung 4.17: Klassendiagramm der Visitor-Klassen

In jedem Categorisation-Objekt kann genau ein Category-Objekt markiert werden. Dadurch kann die Categorisation-Klasse dazu verwendet werden, um die Informationen über den Zelltyp zu speichern, der modelliert werden soll.

In Abb. 4.19 ist ein Klassendiagramm aufgeführt, das den Zusammenhang zwischen der Catgorisation-, der Category- und der ComponentVisitor-Klasse verdeutlicht.

4.3 Serialisierung und Erweiterbarkeit

Ein wichtiger Aspekt, der bei der Entwicklung eines Editors für computergrafische Zellmodelle berücksichtigt werden muss, ist die Notwendigkeit der Bereitstellung eines Mechanismus, der es erlaubt, die Modelle abspeichern und abgespeicherte Modelle einladen zu können.

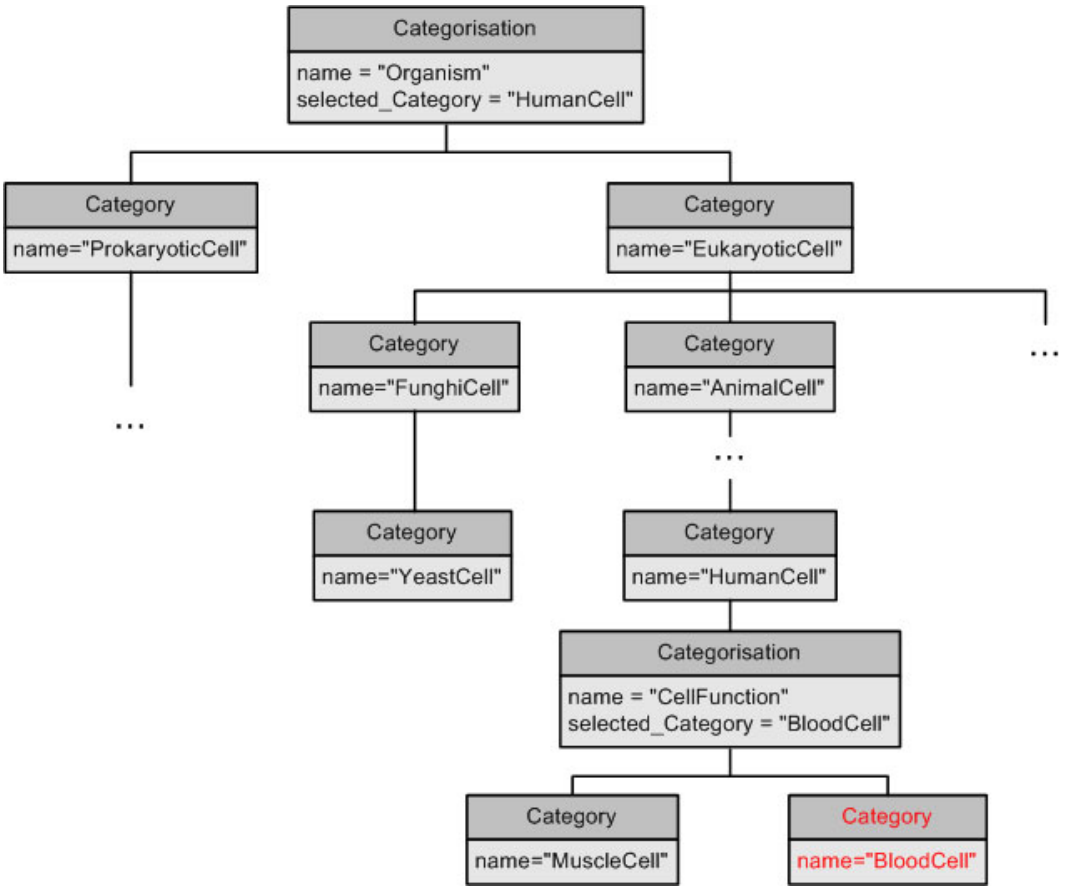

Abbildung 4.18: Beschreibung der Klassifizierung aus Abbildung 4.10 durch Categorisation- und Category-Objekte

Eine weiterer Aspekt betrifft die Erweiterbarkeit des Zellmodells. Es müssen bestimmte Richtlinien entwickelt werden, die bei der Neuimplementierung von Klassen berücksichtigt werden sollten.

In diesem Kapitel werden die Serialisierung und die Erweiterbareit des Zellmodells behandelt. Zuerst wird jedoch eine neue Klasse vorgestellt, die die Erweiterbarkeit vereinfacht und bei der Realisierung der Serialisierung eine entscheidende Rolle spielt.

4.3.1 Die Klasse ComponentLibrary

Der grundlegende Gedanke, die Klasse ComponentLibrary zu entwickeln, lag darin begründet, eine zentrale Instanz festzulegen, die alle bei der Modellierung einer Zelle benötigten Klassen verwaltet. Zu diesen Klassen zählen die zelltypabhängigen Visitoren und die konkreten Bausteinklassen.

4.3.1.1 Verwalten der Bausteine

Die ComponentLibrary dient als Fabrik für Baustein-Objekte, wobei intern zwischen normalen und Border-Bausteinen unterschieden wird. Damit ein Baustein von der Fabrik erzeugt werden kann, muss er zuerst registriert werden. Dazu wird der ComponentLibrary ein Zeiger auf ein prototypisches Objekt der entsprechenden Baustein-Klasse übergeben. Dieser Zeiger wird nun unter dem Namen der Baustein-Klasse in einer Liste gespeichert.

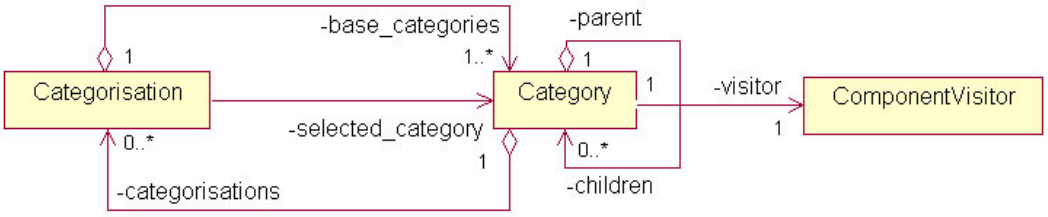

Abbildung 4.19: Klassendiagramm der Categorisation- und der Category-Klasse

Die ComponentFactory erzeugt auf Anforderung, unter Benennung des Klassennamens, einen neuen Baustein. Für diesen Zweck kann aber nicht einfach nur der gespeicherte Zeiger zurückgegeben werden, da dadurch kein neues Objekt erzeugt wird. Aus dem verzeigerten Objekt muss ein Abbild erstellt werden, das dann als neuer Baustein ausgegeben werden kann.

Zur Realisierung dieses Vorgangs wird das in 2.3.2.2 beschrieben Prototype-Pattern eingesetzt. Dafür müssen von jeder Baustein-Klasse eine clone-Funktion und ein geeigneter Copy-Konstruktor bereitgestellt werden. Wird die clone-Funktion eines Baustein-Objektes ausgeführt, erstellt das Objekt eine neue Instanz der eigenen konkreten Klasse, initialisiert diese Instanz mit seinen eigenen Werten und gibt einen Zeiger auf das geklonte Objekt zurück.

4.3.1.2 Verwalten der Kategorien

Bei der Erzeugung der Bausteine muss der Typ der modellierten Zelle berücksichtigt werden. Die möglichen Zelltypen werden durch ein Categorisation-Objekt beschrieben, dieses Objekt wird nun auch in der Klasse ComponentLibrary gespeichert.

Innerhalb des Categorisation-Objektes können Kategorien markiert werden. Die selektierten Kategorien beschreiben den zu modellierenden Typ. Beim Erzeugen eines Bausteins durch die ComponentLibrary kann der ComponentVisitor des selektierten Typs ermittelt werden und die geklonten Bausteine vor der Rückgabe zelltypspezifisch initialisiert werden.

4.3.1.3 Verwalten der Shape-Klassen

Zur Erzeugung von Shape-Objekten wird auf die gleiche Technik wie bei der Erzeugung von Baustein-Objekten zurückgegriffen. Dafür wurde eine Klasse ShapeLibrary entwickelt, die die Prototypen enthält und für die Shape-Klasse eine clone-Funktion und ein Copy-Konstruktor implementiert. Die Verwaltung der ShapeLibrary wird von der ComponentLibrary übernommen (siehe Abb. 4.20).

4.3.2 Serialisierung

Als Serialisierung wird das Abbilden von Variablen- und Objektdaten auf einen zweidimensionalen Bit-Stream bezeichnet. Dieser Bit-Stream wird zur Aufbewahrung und Wiederverwendbarkeit von Daten in einer Datei auf einem Speichermedium abgelegt. Die umgekehrte Vorgehensweise, das Wiederherstellen der serialisierten Datenstrukturen aus einem Bit-Stream, bzw. aus einer Datei, wird als Deserialisierung bezeichnet.

Für die Serialisierung der Daten müssen folgende Sachverhalte beachtet werden:

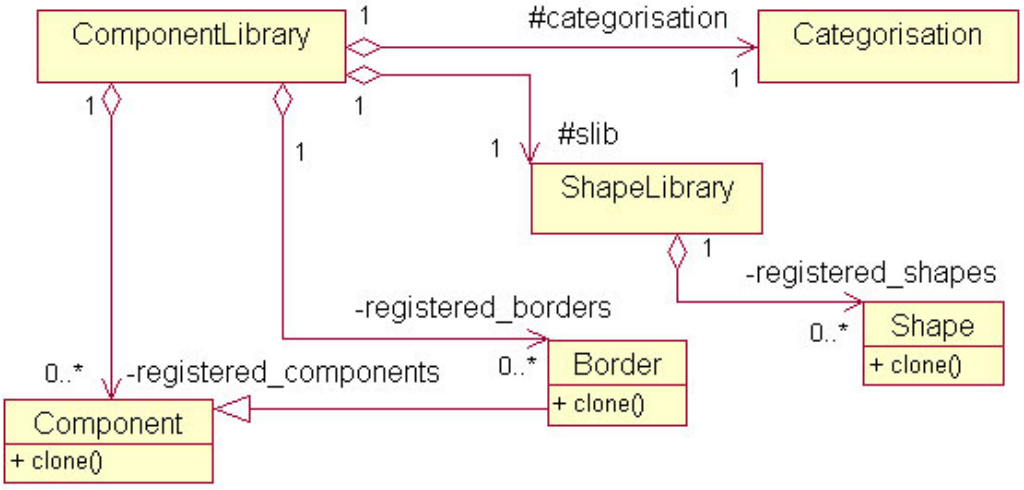

Abbildung 4.20: Klassendiagramm der ComponentLibrary-Klasse

- Primitive Datentypen und Strings lassen sich vergleichsweise einfach serialisieren. Nach dem Schreiben einer Variablen wird ein Leerzeichen angefügt, das das Ende des Variablenwertes markiert. Sollen Strings serialisiert werden, die eventuell ein Leerzeichen enthalten, muss ein anderes Zeichen, das in keinen der Strings vorhanden sein wird, zur Markierung des Stringendes verwendet werden.

- Die Schreib- und die dazugehörigen Leseoperationen müssen aufeinander abgestimmt sein. Die Deserialisierung von Variablen erfolgt in derselben Reihenfolge, in der die Serialisierung der Daten durchgeführt wurde.

- Objekte steuern ihre Serialisierung selbst. Serialisierbare Objekte implementieren eine serialize- und eine deserialize-Funktion. Bei der Objektdeserialisierung müssen zwei Fälle unterschieden werden:

  – Soll ein Objekt serialisiert werden, dessen konkreter Klassentyp bekannt ist, wird zuerst ein neues Objekt der konkreten Klasse des serialisierten Objektes erstellt und anschließend die Deserialisierungs-Funktion des Objektes ausgeführt. Zur Serialisierung des View-Objekt der Component-Klasse kann z. B. auf diese Technik zurückgegriffen werden.

  – Ist die konkrete Klasse des Objektes nicht bekannt, dann ist die Deserialisierung bedeutend schwieriger. Durch Mitspeichern des Klassennamens kann bei der Deserialisierung zumindest der konkrete Typ der Klasse ermittelt werden. Nun muss allerdings eine zentrale Instanz vorhanden sein, durch die über Angabe des Klassennamens ein neues Objekt der Klasse dynamisch erstellt werden kann. Die Deserialisierung erfolgt dann äquivalent wie bei den Objekten mit bekanntem Klassentyp.
    Component- oder Shape-Objekte sind im allgemeinen nur durch ihre abstrakte Klasse bekannt. Für diese Klassen stehen mit der Component- und der ShapeLibrary aber auch zentrale Instanzen zum dynamischen Neuinitialisieren von Objekten durch Angabe des Klassennamens zur Verfügung, die bei der Deserialisierung verwendet werden.

4.3.3 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.

4.3.3.1 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 bereit gestellt. 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.

4.3.3.2 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.

4.3.3.3 Richtlinien zur Erweiterbarkeit

Das in diesem Kapitel 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.

Implementierung neuer Baustein-Klassen

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 wird 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
- ein serialize- und eine deserialize-Funktion, falls der Baustein Attribute besitzt, die bei der Serialisierung mitgeschrieben werden müssen

Gegebenenfalls muss eine neue bausteinspezifischer ManagerKlasse entwickelt werden.

Damit der Baustein bei der Modellierung verwendet werden kann, muss er zum einen in der CellLibrary hinzugefügt werden und zum andern müssen die Visitor-Klassen entsprechend angepasst werden.

Erweitern der modellierbaren Zelltypen

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 Entwicklung der Benutzeroberfläche des 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.

5.1 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 grafischen 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 ihrer bereitgestellten Modellierungsmethoden individuell bearbeiten.
- Hat der Benutzer den Modellierungsprozess fürs Erste abgeschlossen, speichert er das erstellte Projekt ab.

Anwendungsfälle

Ausgehend von dem erdachten Modellierungsprozess werden nun Anwendungsfälle aufgelistet, deren Durchführung über die implementierte grafische Benutzerschnittstelle ermöglicht werden muss. In Abb. 5.1 ist ein für den Zelleditor entworfenes UML-UseCase-Diagramm dargestellt — die darin aufgezeigten Anwendungsfälle werden nun näher beschrieben:

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.

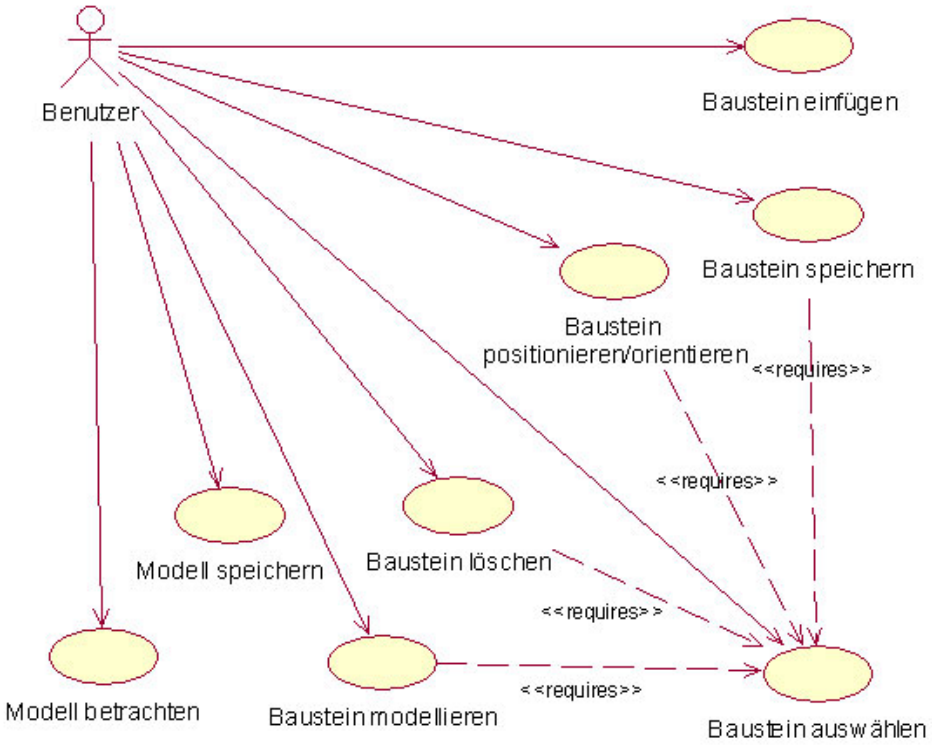

Abbildung 5.1: UML-UseCase-Diagramm zur Beschreibung der Anwendungsfälle des Zellmodellierungsprozesses

- 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 grafischen 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 im Zusammenhang mit den in diesem Abschnitt durchgeführten Betrachtungen eingeführten Begriffe Selektion und Fokussierung sollen nun näher erläutert werden.

5.2 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 grafischen Modell transformiert, gelöscht, gepeichert 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 grafischen Darstellung transformiert werden kann, da es selbst das Wurzelelment des aktuellen Modells bildet und das absolute Koordinatensystem der 3D-Szene definiert.

Fokusierung

Als Fokusierung 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.

5.3 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 Fokusierungen 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 in Abschnitt 2.3.2.4 beschriebene Model-View-Controller-Konzept zurückgegriffen. Das entwickelte Klassenkonzept für die drei Elemente Model, View und Controller wird nun beschrieben (siehe dazu Abb. 5.2).

5.3.1 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 (siehe Abschnitt 2.3.2.4). 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, das von dem Objekt aus dem Modell entfernt wird.

Baustein selektieren Bei der Selektion eines Bausteins gelten folgende Regeln, die sich aus den in 5.2 dargestellten Selektions-Eigenschaften ergeben:

- Befindet sich das ausgewählte Element unmittelbar innerhalb des fokussierten Objektes, oder das Element ist selbst im Besitz des Fokus, wird im Modell das gewählte Objekt selektiert.
- Befindet sich das ausgewählte Element außerhalb der fokussierten Komponente, das heißt 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-Objekt 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.

5.3.2 Die verschiedenen 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.

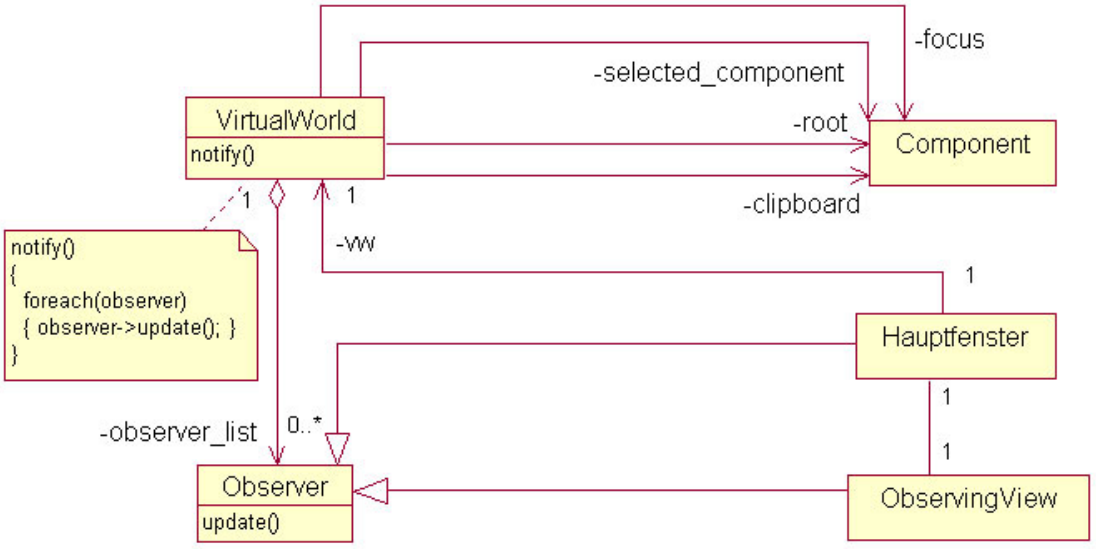

Abbildung 5.2: Das Model-View-Controller-Konzept des Zelleditors

5.3.3 Der Controller

Das Hauptfenster ist im MVC-Konzept des Editors „View“ und „Controller“ zugleich. Die GUI-Elemente des FOX-Toolkits 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 Abb. 5.3 dargestellt.

Als Beispiel für die Notwendigkeit eines solchen Mechanismus sei auf die in Abschnitt 5.5.5 beschriebene Notwendigkeit der Fokussierung der Blickrichtung auf einen Baustein nach der Selektion über den Hierarchie-Browser hingewiesen.

5.4 Windowssystem-abhängige Elemente der Bausteine

Die Baustein-Klassen werden frei von GUI-Toolkit-abhängigem Code gehalten. Dies ist erforderlich, da noch keine Programme zur Weiterverwendung der generierten Zellmodelle entwickelt wurden und bei deren zukünftiger Implementierung die Möglichkeit offen gehalten werden sollte, auf andere GUI-Toolkits als den FOX-Toolkit, der bei dem hier vorgestellten Projekt verwendet wurde, zurückgreifen zu können.

Innerhalb des Modellierungstools gibt es allerdings Windowssystem-abhängige GUI-Elemente für die unterschiedlichen Bausteintypen. Jeder Baustein besitzt z. B. an-

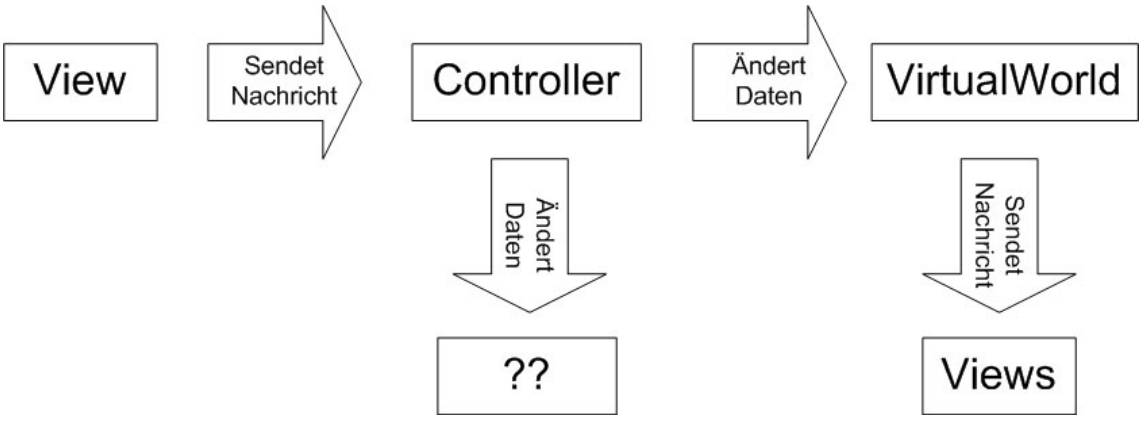

Abbildung 5.3: Nachrichtenübertragung View-Controller-VirtualWorld

dere Editierfunktionen, die durch unterschiedliche Dialoge, Bedienfelder oder andere Windows-Komponenten bereitgestellt werden.

Es wird nun ein geeigneter Ansatz beschrieben, durch den es möglich, ist die einzelnen Bausteine um die Windowssystem-spezifischen Elemente zu erweitern.

5.4.1 FXComponent

Für die Erweiterung der Bausteine um Windows-Bedienelemente unter der Bedingung, dass die Basiklassen frei von Windowssystem-spezifischem Code bleiben sollen kommt als erste Möglichkeit eine Funktionserweiterung durch Vererbung in Betracht. Für jedem Baustein wird dabei eine konkrete Klasse entwickelt, die den Baustein um Funktionen erweitert, durch die die GUI-Elemente in die Benutzeroberfläche eingebunden werden können. Dieser Mechanismus wird in Abb. 5.4 dargestellt.

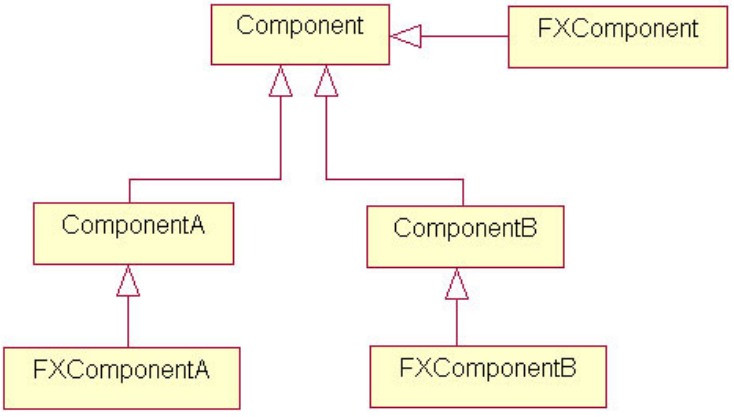

Abbildung 5.4:

Nachteil bei dieser Methode ist, dass die konkreten Windows-Bausteinklassen nicht in einem eigenen Vererbungsbaum liegen, sondern für jeden Baustein eine eigenständige Klasse definiert werden muss. Außerdem existiert keine Basisklasse für

die Windows-Bausteine, wodurch die einzelnen konkreten Klassen der Windows-Bausteine im Programm bekannt sein müssen.

Ein weiterer Ansatz ermöglicht die Trennung der Vererbungsbäume für die normalen und die Windowssystem-abhängigen Bausteinklassen. Dafür wird die Methode der Vererbung durch die Methode der Objektkomposition ersetzt: Jeder Windows-Baustein kann ein Objekt der Baustein-Klasse, die er erweitern soll, aufnehmen und besitzt als Hüllklasse die Möglichkeit im Rahmen der öffentlichen Zugriffsrechte auf Funktionen und Attribute des umhüllten Objektes zuzugreifen. Der Klassenaufbau dieser Art der Funktionalitätserweiterung ist in Abb. 5.5 dargestellt.

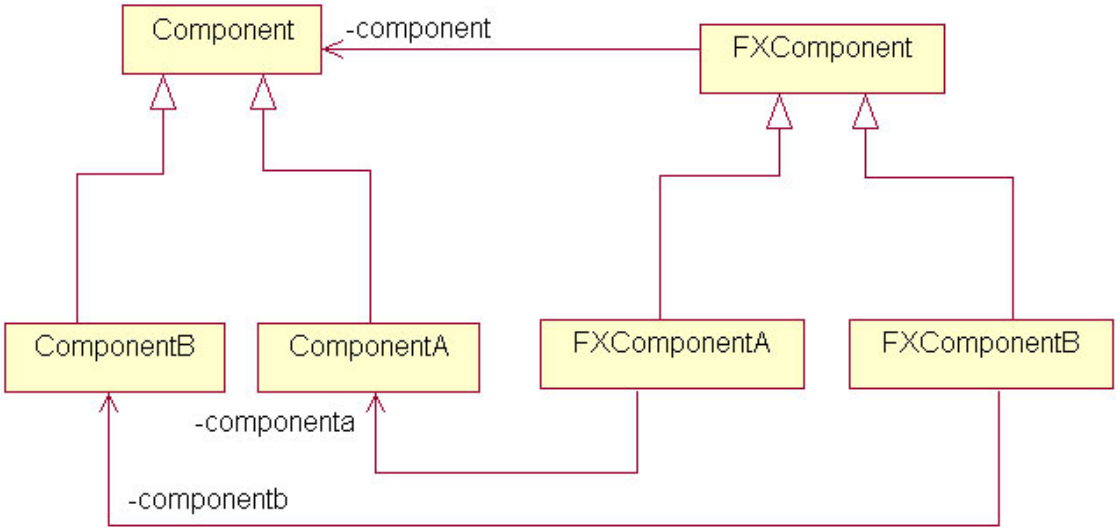

Abbildung 5.5:

Die Klasse FXComponent ist die Basisklasse des Vererbungsbaums der Windowssystem-abhängigen Bausteinklassen. Für diese Klasse werden sechs Funktionen definiert, durch die folgende GUI-Elemente erstellt und in die Benutzeroberfläche eingefügt werden können:

- ein Icon, das z. B. für Schaltflächen eingesetzt werden kann
- einen Dialog, der gestartet wird, wenn ein Baustein neu erstellt wird
- ein Bedienelement zum Verändern von bausteinspezifischen Eigenschaften
- ein Bedienelement, über das die Darstellungseigenschaften eines Bausteins verändert werden können
- ein Bedienelement, das zusätzliche Tools zur Modellierung eines Bausteins zur Verfügung stellt
- ein Bedienelement, in dem Informationen über die Bausteine aufgeführt sind und Hilfestellungen angeboten werden.

Das Icon wird bei der Konstruktion eines Objektes der FXComponent-Basisklasse übergeben, wodurch es möglich ist, die Basisklasse als Windows-spezifischen Baustein für mehrere Bausteine mit jeweils unterschiedlichem Icon zu verwenden.

Das einzige GUI-Element, das die FXComponent-Klasse standardmäßig bereitstellt ist ein Dialogfenster, das beim Erstellen eines Bausteins mit begrenzter Struktur geöffnet wird. Bei diesen Bausteinen ist es wichtig, dass, bevor der Baustein in Unterbausteinen modelliert werden kann, bereits die Außengrenzen festgelegt worden sind. Soll bei einem Zellkern z. B. ein Kernkörperchen eingefügt werden, müssen zuerst die Membranen vorhanden sein, da sonst nicht klar ist, in welchem Bereich die Grenzen des Zellkerns liegen.

Ausgehend von den im ComponentManagers des zu erstellenden Bausteins festgelegten BorderRestrictions wird zuerst für jede Begrenzungsschicht ein neuer Border-Baustein in die Komponente eingefügt. Für jeden dieser neueingefügten Border-Bausteine wird im Dialogfenster eine Karteikarte angezeigt.

Im Normalfall enthält diese Karteikarte jeweils zwei Elemente. Über eine Button-Group aus Radio-Buttons kann die Form des entsprechende Borders gewählt werden. Welche Formen zur Auswahl stehen, ist abhängig von den im BorderManager des Bausteins festgelegten ShapeRestrictions. Unterhalb der Button-Group ist ein Bedienelement vorhanden, mit dem Einstellungen an den Shape-Eigenschaften durchgeführt werden können. Für jeden Shape-Parameter ist ein Scrollfeld vorhanden. Die Standard-Parameter ermittelt das Programm aus den Durchschnittswerten aus den Parametereinschränkungen der ShapeRestriction der gewählten Form.

Ist das Border-Objekt abhängig von der Form seines Vorgängers, dann präsentiert sich das Bedienfeld in einer anderen Form. Über ein Auswahlfeld kann der Abstand zum Vorgänger bestimmt werden, über einen Button besteht die Möglichkeit die Abhängigkeit vom Vorgänger-Objekt aufzuheben. Hierbei ist darauf zu achten, dass dieser Vorgang nicht umkehrbar ist. In Abb. 5.6 ist ein Dialogfenster zum Neuerstellen von begrenzten Bausteinen abgebildet.

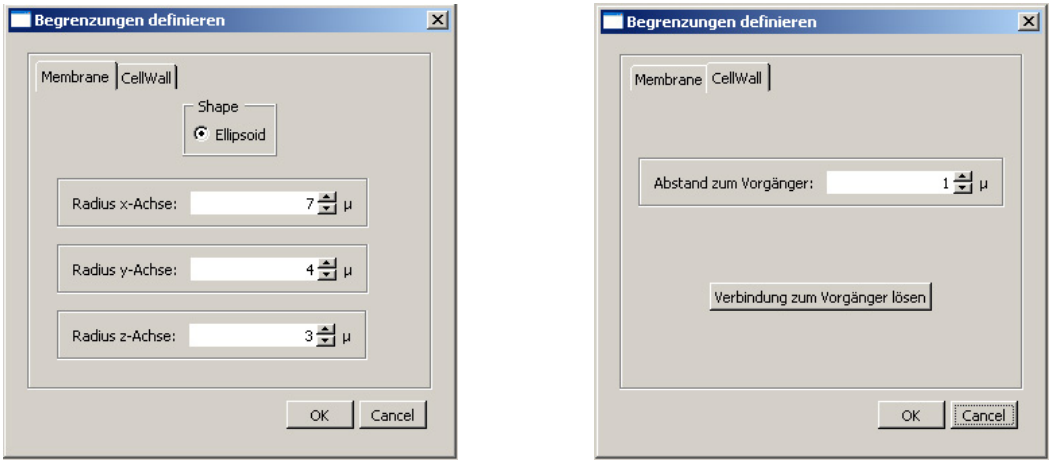

Abbildung 5.6: Dialog zum Neuerstellen eines Zell-Bausteins

Die sechs durch die FXComponent-Klasse definierten Bedienelemente können durch erweiterte FXComponent-Klassen beliebig gestaltet werden. Diese Elemente werden im Normalfall je nach Bedarf automatisch in die Benutzeroberfläche eingefügt.

Für Border-Bausteine wurde eine eigene FXComponent-Klasse entwickelt, deren Aufbau im nächsten Abschnitt beschrieben wird.

5.4.2 FXBorder

Die GUI-System-spezifische Klasse für die Border-Bausteine ist die Klasse FXBorder, die das Bedienelement für die Darstellungseigenschaften von Border-Objekten neu definiert. Da Border-Objekte bei der Neuinitialisierung von begrenzten Bausteinen automatisch miterzeugt werden und im Modell nicht gelöscht werden können, wird kein Erstellungsdialog bereitgestellt.

Das Bedienelementes für die Darstellungseigenschaften eines Border-Objektes ist so aufgebaut wie die entsprechende Karteikarte des Baustein-Objektes im Entstehungsdialog des begrenzenden Bausteins, mit der Ausnahme dass die Form nicht mehr neubestimmt werden kann, sondern nur die Parameter eingestellt werden können.

Die vorgestellten Klassen FXComponent und FXBorder müssen entsprechend den Anforderungen der Bausteine, für die sie GUI-Elemente bereitstellen sollen, erweitert werden. Damit die Windowssystem-abhängigen Klassen übersichtlich administriert werden können wurde die Klasse FXComponentLibrary entwickelt.

5.4.3 FXComponentLibrary

Wie die normalen Baustein-Klassen werden auch die Windowssystem-spezifischen Bausteinklassen in einer Bibliothek verwaltet, die über die Klasse FXComponentLibrary bereitgestellt wird. Da für alle Baustein-Objekte einer bestimmten Baustein-Klasse ein und dasselbe FXComponent-Objekt verwendet werden kann, muss für die FXComponent-Klassen keine clone-Funktion implementiert werden.

Für FXComponentLibrary-Klasse existiert eine spezialisierte Klasse für die Zellmodellierung, die Klasse FXCellLibrary. Jeder konkreten Component-Klasse die Windowssystem-abhängige Eigenschaften besitzt, muss bei der Konstruktion eines Objektes der Klasse FXCellLibrary ein entsprechendes FXComponent-Objekt zugewiesen werden.

5.5 Vorstellung der Benutzeroberfläche

In diesem Abschnitt werden die einzelnen Elemente der entwickelten Benutzeroberfläche des Zellmodellierungswerkzeuges vorgestellt, dazu wird zunächst der Aufbau des Hauptfensters erläutert.

5.5.1 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 Karteikartenbedienelement und ein Hierarchie-Browser in das Hauptfenster eingefügt (siehe Abbildung 5.7).

Das Hauptfenster dient als Controller für die verschiedenen Views und verwaltet diejenigen Werkzeuge, die nicht als View implementiert sind und somit nicht über

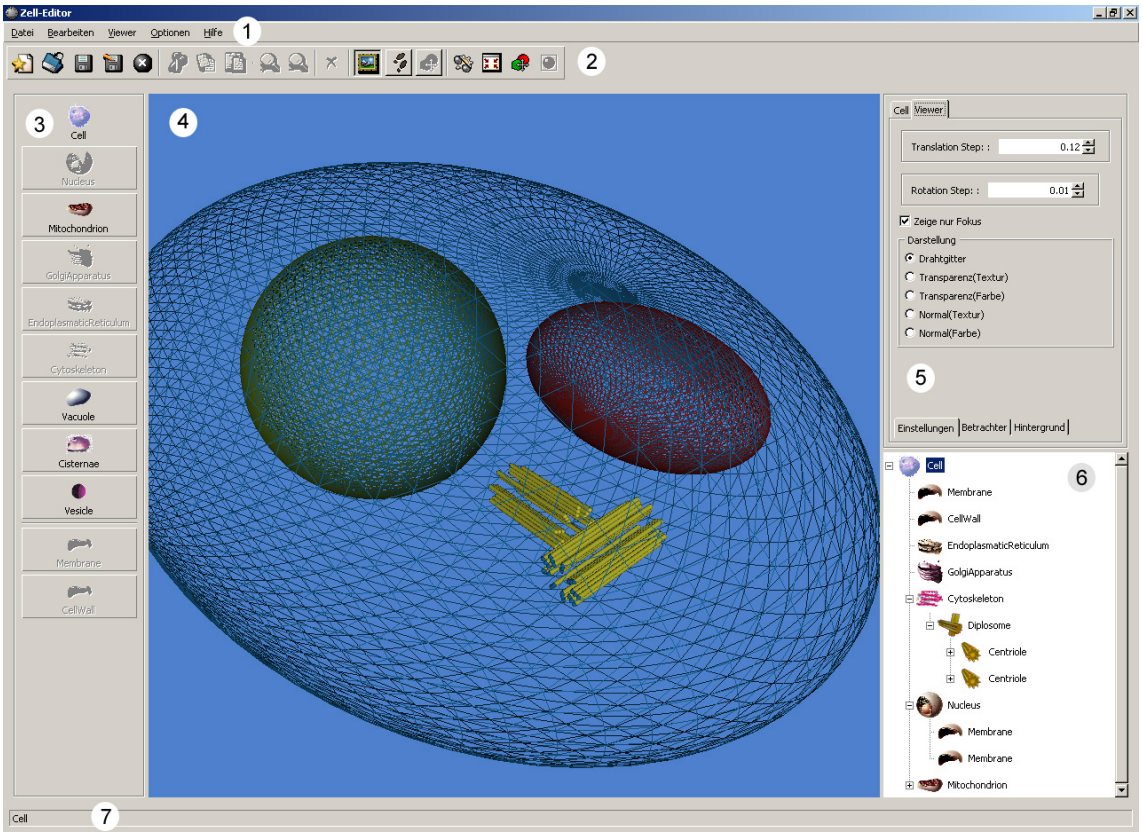

Abbildung 5.7: Das Hauptfenster des Zelleditors; *1 - Menüleiste, 2 - Werkzeugleiste, 3 - Baustein-Buttonleiste, 4 - Szenenbetrachter, 5 - Karteikartenelement, 6 - Hierarchie-Browser, 7 - Statusleiste*

Änderungen des Zellmodells informiert werden. Zu diesen Werkzeugen zählen die Menü- und die Werkzeugleiste, die Statusleiste und das Karteikarten-Bedienelement.

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.

5.5.1.1 Menü- und Werkzeugleiste

Das Hauptfenster enthält eine bei grafischen Benutzerschnittstellen übliche Menü- und Werkzeugleiste, über die bestimmte Operationen durchgeführt werden können. Der Aufbau dieser Leisten kann von anderen Werkzeugen erweitert werden.

5.5.1.2 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.

5.5.1.3 Karteikartenelement

Das Karteikartenelement ermöglicht es, mehrere Einstellungsbedienfelder auf einer kleinen Fläche zu vereinen. Bislang besitzt das Karteikartenbedienfeld zwei Karteikarten, eine für das in Abschnitt 5.5.4 beschriebene Bedienfeld für bausteinspezifische Eigenschaften und eine für Einstellungen an dem im nächsten Kapitel vorgestellten Szenenbetrachter.

5.5.2 Der Szenenbetrachter

Um das 3D-Modell der Zelle in seinem grafischen 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 3D-Szene zu erkunden und 3D-Objekte innerhalb der Szene positionieren oder orientieren zu können.

In kommerziellen 3D-Modellierungstools, wie z. B. 3D-Studio Max, 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 VirtualWorldViewer definiert einen GUI-unabhängigen Betrachter, die Klasse FXVirtualWorldViewer 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.

5.5.2.1 Die Klasse Viewer - ein virtueller Betrachter

Für den Szenenbetrachter wird ein virtueller Betrachter beschrieben, der sich innerhalb der 3D-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. In Abbildung 5.8 wird die Definition des Betrachters durch die drei Vektoren dargestellt.

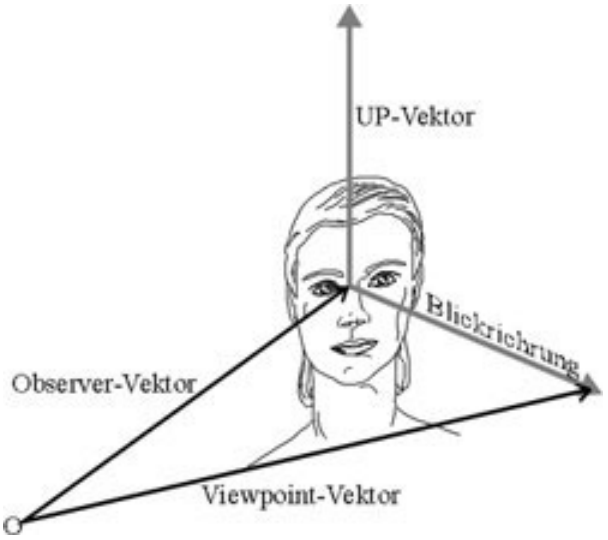

Abbildung 5.8: Vektoren zum definieren des virtuellen Betrachters[VoMü00]

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 Orientierungs- 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 wird 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 xy-Ebene in Blickrichtung des Betrachters aufgespannt ist. Diese Projektion wird durch vier Parameter festgelegt:

near Der Abstand einer als Near-Clipping-Pane bezeichneten Ebene

far Der Abstand zu einer als Far-Clipping-Pane bezeichneten Ebene

fovy Der Winkel des durch den Betrachter abgedeckten Sichtfeldes

aspectRatio Das Breite/Höhe-Verhältnis der Projektionsfläche

Durch die Near- und Far-Clipping-Ebenen wird ein Pyramidenstumpf definiert (siehe Abb. 5.9), 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.

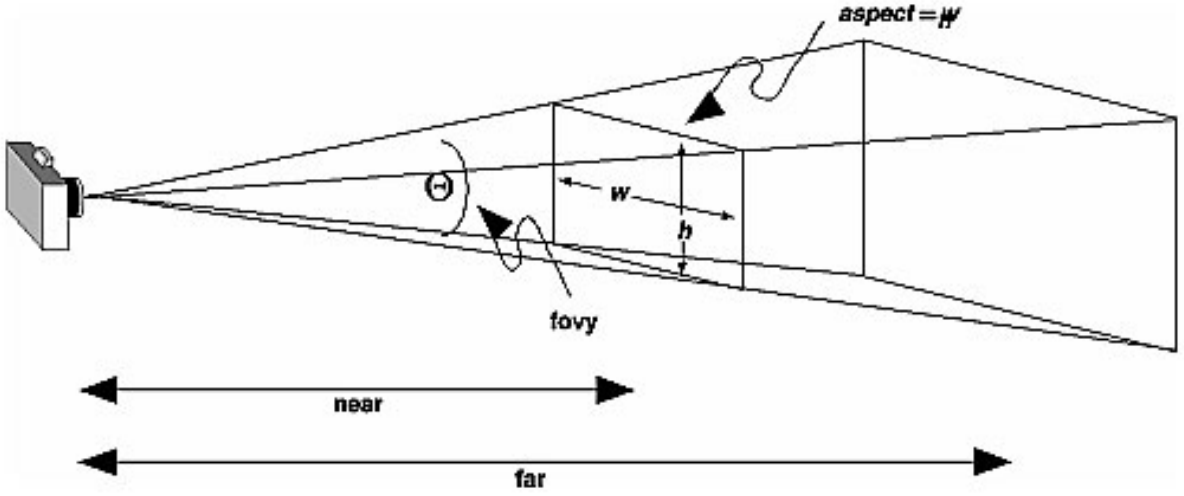

Abbildung 5.9: Der Pyramidenstumpf einer perspektivischen Projektion (gluPerspective)[WoND99]

5.5.2.2 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 restlichen 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 3D-Szene betrachtet werden soll. Die Maussteuerung dieses EventHandlers ist so ausgerichtet, dass dem Benutzer das Gefühl vermittelt wird, als würde er 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 gegengesetzter Richtung der 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 3D-Szene eingesetzt. Die Maussteuerung entspricht der Steuerung des SceneExaminers, 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.

5.5.2.3 Die Klasse VirtualWorldViewer

Es wurde zunächst eine Klasse VirtualWorldViewer entwickelt, die einen GUI-unabhängigen Viewport mit Interaktionsmöglichkeiten für eine durch ein VirtualWorld-Objekt repräsentierte 3D-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 EventHandler, der beliebig ausgetauscht werden kann.

Weiterhin werden einige weitere wichtige Funktionen zur Verfügung gestellt:

Baustein anschauen Über diese Funktion wird der Betrachter so gedreht, dass derMittelpunkt 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.

5.5.2.4 Die Klasse FXVirtualWorldViewer

Die VirtualWorldViewer-Klasse stellt zusammen mit den EventHandler-Klassen und der Viewer-Klasse ein Grundgerüst eines Szenenbetrachters dar. Um Benutzerinteraktionen zu ermöglichen, muss diese Grundstruktur noch in den Windowing-Kontext eingebunden werden

Zu diesem Zweck wurde die Klasse FXVirtualWorldViewer entwickelt. Diese Klasse ist eine Erweiterung der Klasse FXGLCanvas, die ein OpenGL-Rendering-Fenster unter FOX beschreibt und gleichzeitig eine Erweiterung der VirtualWorldViewer-Klasse. Die Klasse empfängt FOX-Maus- und Tatsaturevents und leitet sie an den aktuellen EventHandler weiter. Auf der anderen Seite dient sie als Kommunikationsschnittstelle zwischen dem EventHandler und dem VirtualWorld-Objekt, um dort Selektionen auszuführen. Die Zusammenhänge zwischen den Klassen der entwickelten Elemente des Szenenbetrachters wird in Abb. 5.10 dargestellt.

Weiterhin werden von der Klasse FXVirtualworldViewer verschiedene Bedienelemente zum Ändern von Einstellungen des Szenenbetrachters zur Verfügung gestellt, die beliebig im Zelleditor platziert werden können:

- Um zwischen den verschiedenen EventHandlern wechseln zu können, werden für den SceneExaminer, den SceneExplorer und den ObjectManipulator Buttons zur Verfügung gestellt. Durch Betätigen eines solchen Buttons wird ein neuer EventHandler ausgewählt, der gewählte Button wird im GUI entsprechend als gedrückt gezeichnet.
  Eine Besonderheit ist beim Aktivieren des ObjectManipulators zu berücksichtigen: Ist das selektierte Element, das zur Manipulation vorgesehen ist, auch das zur Zeit fokussierte Element, dann muss zuerst der Fokus auf den Vaterbaustein übergehen, damit Transformationen an dem Baustein durchgeführt werden können. Für das Wurzelelement kann der ObjectManipulator nicht aktiviert werden, da dieses Element keinen Vater-Baustein besitzt, in dessen Kontext er transformiert werden könnte.

- Für die Funktionen „Baustein ansehen“, „Baustein-Zentrum aufsuchen“, „Baustein zentrieren“ und „Baustein in den Viewport bewegen“ werden ebenfalls

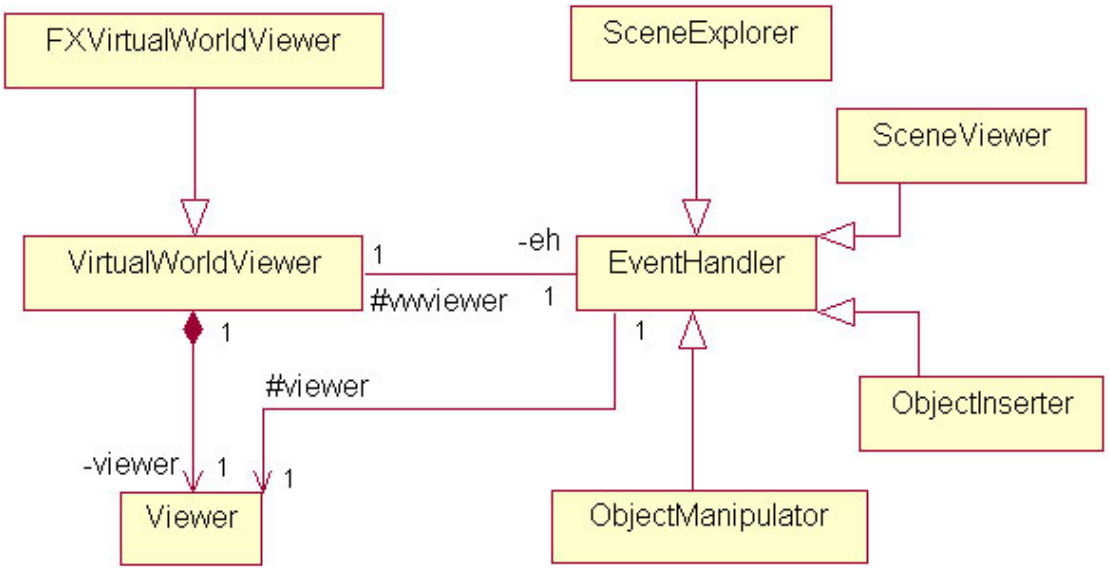

Abbildung 5.10: Klassenmodell des Szenenbetrachters

Buttons bereitgestellt. Der Button zum Verschieben des Betrachters in den Baustein-Mittelpunkt ist nur dann aktiv, wenn der selektierte Baustein eine begrenzte Struktur aufweist und somit auch eine Innenansicht besitzt.

- Um verschiedene Darstellungseigenschaften zu definieren, wird ein entsprechendes Bedienelement zur Verfügung gestellt, das das Wechseln zwischen unterschiedlichen Darstellungsarten ermöglicht:

  Drahtgitter Alle Bausteine werden als Drahtgittermodell dargestellt. Diese Darstellungsart ist die Default-Einstellung des Szenenbetrachters.

  Transparenz (Textur) Die Bausteine werden transparent mit Texturen gezeichnet.

  Transparenz (Farbe) Die Bausteine werden transparent ohne Texturen gezeichnet.

  Normal (Textur) Die Bausteine werden in der normalen Flächendarstellung mit Texturen gezeichnet.

  Normal (Farbe) Die Bausteine werden in der normalen Flächendarstellung ohne Texturen gezeichnet.

  In diesem Bedienelement befinden sich außerdem Scrollelemente, durch die die Schrittweite der Bewegung des Betrachters und der Rotationswinkel für Drehbewegungen eingestellt werden können. Diese Werte werden bei Mausrad- und Tastaturinteraktionen berücksichtigt und spielen für die übrigen Maus-Manipulationen keine Rolle.

- Ein weiteres Bedienelement bietet die Funktionen, die Hintergrundfarbe der gerenderten Szene festzulegen.

Die Buttons des FXVirtualWorldViewers werden in die Werkzeugleiste des Hauptfensters eingebunden. Die drei Bedienelemente werden als Karteikarten innerhalb der Karteikarte Viewer des Karteikartenelementes des Hauptfensters eingefügt.

5.5.3 Baustein-Buttonleiste

Die Baustein-Buttonleiste wird verwendet, um Bausteine in das Modell einzufügen. Sie besteht aus mehreren Buttons, von denen jeder einen bestimmten Baustein repräsentiert. Nach Betätigung des Buttons wird eine Einfügeoperation durchgeführt.

Die Klasse FXComponentToolbar beschreibt eine solche Buttonleiste. Der Aufbau der Leiste hängt von dem fokussierten Objekt ab, weshalb sie als Observer-Objekt des Virtualworld-Objektes registriert ist. Änderungen des fokussierten Elementes erzwingen eine Neuinitialisierung der Buttonleiste.

Bei den bereitgestellten Buttons muss zwischen Border- und normalen Bausteinen unterschieden werden, da die normalen Bausteine nur durch ihren Namen identifiziert werden können, für die Border-Bausteine zusätzlich allerdings noch der Index der Schicht, in der der Border-Baustein eingefügt werden soll, mitgespeichert werden muss.

Bei der Initialisierung der Buttonleiste werden die Richtlinien des ComponentMangers berücksichtigt:

- Für alle Bausteine, die in den CompositionRestrictions registriert sind wird ein Button eingefügt. Falls für einen Baustein schon die maximal erlaubte Anzahl erreicht ist, wird der Button deaktiviert.
- Für alle Bausteine, die in den BorderRestrictions registriert sind wird ein Button eingefügt. Falls ein Border schon gesetzt ist, wird der Button deaktiviert.

In Abb. 5.11 ist eine Baustein-Leiste für die Elemente des Zytoskeletts dargesellt.

5.5.4 Bedienfeld für die bausteinspezifischen Eigenschaften

Innerhalb des Karteikartenelements des Hauptfensters ist eine Karteikarte für Bedienfelder zur Manipulation von bausteinspezifischen Eigenschaften reserviert. In dieser Karteikarte wird ein GUI-Element der Klasse FXComponentPropertyPanel eingefügt. Dieses besitzt wiederum ein eigenes Karteikartenelement, das vier Karteikarten, für die vier durch die FXComponent-Klasse definierten Bedienelemente bereitstellt.

Der Aufbau dieser Karteikarten hängt von dem aktuell selektierten Baustein ab. Aus der FXComponentLibrary wird die entsprechende FXComponent-Klasse für den selektierten Baustein angefordert, die entsprechenden Bedienelemente dieser Klasse werden anschließend in das Karteikartenelement eingefügt.

5.5.5 Hierarchie-Browser

Der Hierarchie-Browser stellt die komplette Bausteinhierarchie des Zellmodells in einer Baumansicht dar, ähnlich wie es aus der Darstellung der Struktur des Dateisystems in Verzeichnislisten bekannt ist.

Der Hierarchie-Browser wird durch die Klasse FXSceneGraph beschrieben. Dem Nutzer steht mit dem Hierarchie-Browser ein Werkzeug zur Verfügung, mit dem er den strukturellen Aufbau aller Bausteine des Modells betrachten kann. Außerdem wird ihm die Möglichkeit geboten, über diese Ansicht Komponenten im Modell zu selektieren. Vorteil bei dieser Selektionsmethode ist, dass die zu selektierende Komponente nicht im Blickfeld des Betrachters liegen muss, dadurch lässt sich sehr schnell innerhalb der Szene navigieren und zwischen verschiedenen Modellierungsebenen wechseln.

Da sich nach der Selektion die Komponente nicht unbedingt im Blickfeld des Betrachters befindet, wird nach der Selektion das neu selektierte im Betrachterfenster zentriert. Das Hauptfenster empfängt hierbei eine Selektionsanforderung und führt nach Identifikation des Hierarchie-Browsers als Sender der Anforderung über den Szenenbetrachter zusätzlich eine Zentrierung des neuselektierten Objektes durch.

In Abb. 5.12 ist ein Hierarchie-Browser des Zelleditors dargestellt.

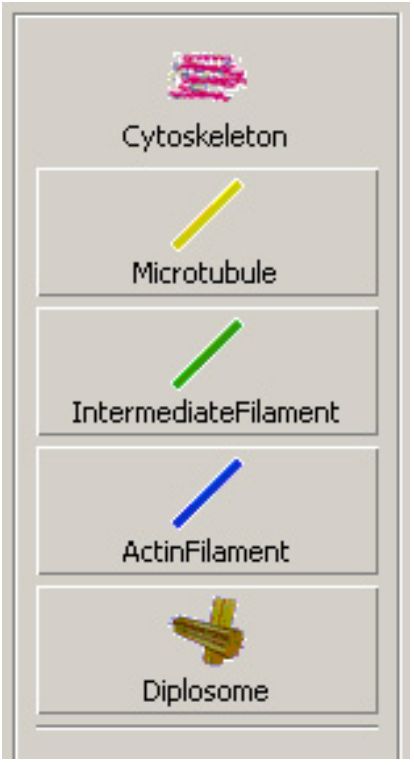

Abbildung 5.11: Die Baustein-Werkzeugleiste des Zelleditors für das Zytoskelett der Zelle

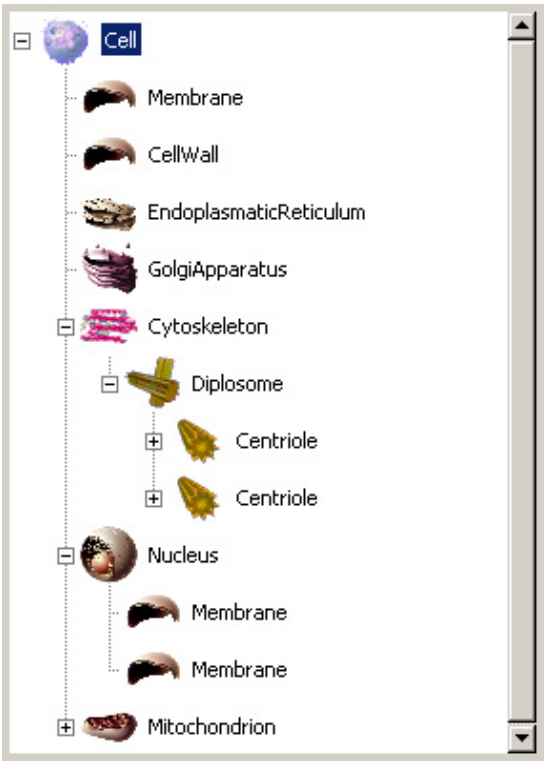

Abbildung 5.12: Hierarchie-Browser des Zelleditors

5.6 Beschreibung einiger Funktionsabläufe

Um den Modellierungsprozess etwas präziser darzustellen und die Zusammenhänge zwischen den einzelnen Elementen des Modellierungswerkzeuges und den Modelldaten zu beschreiben, werden nun einige ausgewählte Funktionsabläufe aufgeführt.

5.6.1 Neues Modell erstellen

Die Anforderung zum Erstellen eines neuen Modells wird über die Menü- oder die Werkzeugleiste initiiert. Das sich öffnende Wizard-Fenster besitzt zwei Seiten, in denen Einstellungen über das zu modellierende Objekt gemacht werden (siehe Abb. 5.13).

In der ersten Seite wird der Zelltyp des zu modellierenden Objektes bestimmt, in der zweiten Seite kann bestimmt werden, welcher Baustein-Typ modelliert werden soll. Da die Zelle in der Programmlogik ebenso ein Baustein darstellt wie alle ihre Unterbausteine, ist es möglich jeden beliebigen Baustein seperat zu modellieren und als eigenes Modell abzuspeichern. Ausnahmen bilden die Border-Bausteine, die nur in Verbindung mit einem Baustein mit begrenzter Struktur modelliert werden können und die komplexen Bausteine.

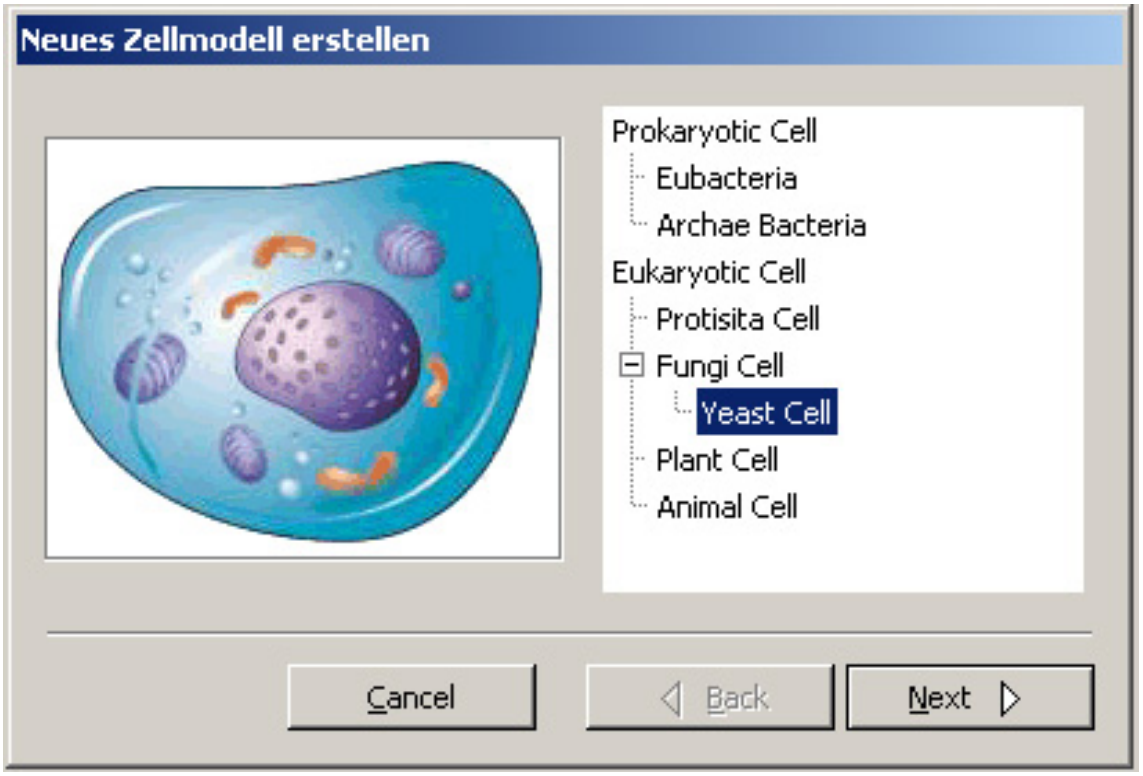

Abbildung 5.13: Dialog zum Starten eines neuen Modellierungsprojektes

5.6.2 Baustein selektieren

Selektierungen können über den Szenenbetrachter durch Klicken auf bestimmte Objekte der Szene bei aktivierten SceneExplorer oder SceneExaminer-EventHandler oder über den Hierarchie-Browser ausgeführt werden.

Nach der Selektion eines Bausteins wird ein neues Bedienelement für die bausteinspezifischen Eigenschaften für den neuen Baustein bestimmt. Erfolgte die Selektierung über den Hierarchie-Browser, dann wird das neuselektierte Objekt im Szenenbetrachter zentriert.

5.6.3 Baustein fokussieren

Fokussierungen können über die Werkzeugleiste oder den Hierarchiebrowser vorgenommen werden. Wird ein Baustein fokussiert, dann wird im Szenenbetrachter nur die durch den Baustein definierte 3D-Szene dargestellt. Dies gilt allerdings nicht für komplexe und Border-Bausteine, da diese Elemente nur innerhalb der übergeordneten Modellierungsebene gezeigt werden sollten.

Nach der Fokussierung ändert sich die Baustein-Buttonleiste in Abhängigkeit vom fokussierten Baustein.

5.6.4 Baustein einfügen

Die Einfügeoperation wird durch Betätigen einer Schaltfläche der Baustein-Buttonleiste gestartet. Durch das Hauptfenster wird daraufhin von der CellLibrary ein neuer Baustein angefordert, dieser wird dann an das VirtualWorld-Objekt übergeben, welches letzendlich den neuen Baustein in die Szene des fokussierten Elementes einfügt.

5.6.5 Baustein löschen

Das Löschen eines Bausteins wird über die Menü- oder Werkzeugleiste ausgeführt. Der selektierte Baustein wird nach Betätigen eines Lösch-Kommandos aus dem Modell entfernt. Es können nur normale Component-Bausteine gelöscht werden, Border-Bausteine können nicht gelöscht werden, um zu verhindern, dass ein begrenzter Baustein seine Begrenzungen verliert.

5.6.6 Baustein speichern

Bausteine können über die entsprechenden Kommandos der Menü- und Werkzeugleiste ausgeführt werden. Border-Bausteine können nicht abgespeichert werden, da diese Elemente immer in Beziehung zu einem begrenzten Baustein stehen sollten. Auch das Abspeichern von komplexen Strukturen ist wenig sinnvoll, da die Ausdehnung dieser Strukturen durch den den Modellierungskontext bildenden Baustein definiert wird.

6 Implementierung der Komponentenklassen zur Modellierung einer Hefezelle

Das in Kapitel 4 entwickelte computerbasierte Zellmodell wird in diesem Kapitel um konkrete Klassen erweitert, die zur Modellierung einer Hefezelle herangezogen werden können. Die aufgezeigte Vorgehensweise soll die Weiterentwicklung der Basisklassen am Beispiel der Bereitstellung von Klassen für die Hefezellen-Modellierung darstellen.

6.1 CellLibrary und FXCellLibrary

Die Klassen CellLibrary und FXCellLibrary wurden als Bibliotheken zum Verwalten der für die Zellmodellierung benötigten Komponenten und Informationen entwickelt. Diese Klassen müssen durch die in diesem Kapitel neuimplementierten Klassen ergänzt werden. Zu den neuentwickelten Klassen zählen der im nächsten Abschnitt beschriebene YeastVisitor und die in den darauffolgenden Abschnitten vorgestellten konkreten Shape- und Baustein-Klassen.

6.2 CellVisitor und YeastVisitor

In der Klasse CellVisitor werden Modellierungsrichtlinien verwaltet, die für alle Zelltypen gelten. Diese Klasse wird um eine visit-Funktion für alle in diesem Kapitel vorgestellten Bausteine erweitert.

Für die Hefezelle wurde eine spezielle CellVisitor-Klasse entwickelt, die die Modellierungsrichtlinien für die Bausteine der Hefezelle enthält. Die neue Klasse *YeastVisitor* überschreibt die bausteinabhängigen visit-Funktionen der CellVisitor-Klasse für alle Bausteine, die spezifische Richtlinien für die Modellierung einer Hefezelle besitzen.

Die Modellierungsrichtlinien wurden so gut wie möglich aus den in Abschnitt 2.1.3 des Grundlagenkapitels zusammengestellten Angaben über Form, Größe und Aufbau

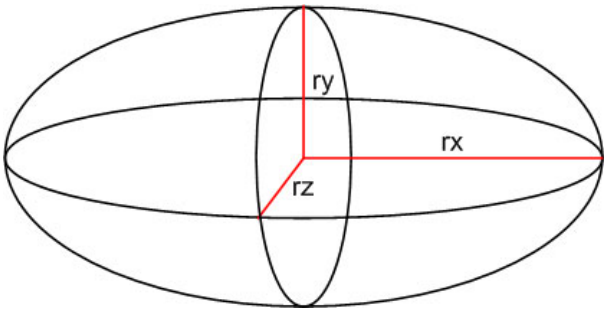

Abbildung 6.1: Ein ellipsoidischer Körper

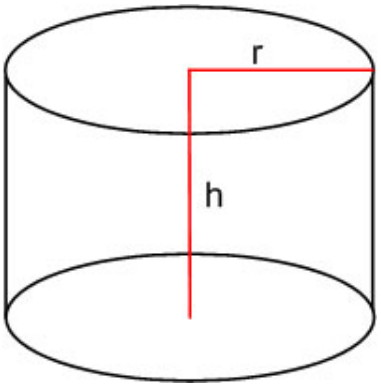

Abbildung 6.2: Ein zylindrischer Körper

der Komponenten und insbesondere aus den in Abschnitt 2.1.4 aufgeführten Informationen über Hefezellen zusammengetragen und nach eigenem Ermessen ergänzt.

Damit der neuentwickelte Visitor beim Modellierungsprozess berücksichtigt wird, muss der durch Categorisation- und Category-Objekte dargestellte Klassifizierungsbaum der CellLibrary um eine Kategory für die Hefezelle erweitert werden, die ein YeastVisitor-Objekt zugewiesen bekommt.

6.3 Shape-Klassen

Es werden nun konkrete Shape-Klassen vorgestellt, die als Elementarobjekte für die Bausteinmodellierung verwendet werden können. Für die 3D-Darstellung aller implementierten Bausteine der Hefezelle werden die drei Grundformen Kugel, Ellipsoid und Zylinder zur Verfügung gestellt. Jeder dieser drei Körper wird durch eine eigene konkrete Shape-Klasse repräsentiert, die nun kurz vorgestellt wird.

Die Klasse Ellipsoid

Zur Darstellung ellipsoidischer Körper wird die Klasse Ellipsoid bereitgestellt. Ein Ellipsoid läßt sich durch drei Parameter beschreiben, die jeweils einen Radius für eine der drei Raumachsen definieren (siehe Abb. 6.1).

Die Klasse Sphere

Eine Kugel wird durch die Klasse Sphere repräsentiert. Die Kugel kann als Spezialform einer Ellipse angesehen werden, die Klasse Sphere wurde deshalb als Spezialisierung der Ellipsoid-Klasse entwickelt. Eine Kugel ist ein Ellipsoid, dessen drei Raumachsen-Radien gleich groß sind und wird somit eindeutig durch einen einzigen Radius-Parameter definiert.

Die Klasse Cylinder

Ein Zylinder wird durch einen Radius und einen Höhenwert eindeutig definiert (siehe Abb. 6.2), die Klasse Cylinder dient zur Beschreibung eines solchen Körpers.

6.4 Bausteinklassen der Hefezelle

Es werden nun die implementierten Bausteinklassen vorgestellt, die zur Modellierung der Hefezelle verwendet werden können. Die in diesem Zusammenhang entwickelten Bausteine sind nicht zelltypspezifisch konzipiert worden und können durchaus auch zur Modellierung von Zellen eines anderen Typs eingesetzt werden, die typischen Eigenarten der Hefezelle werden durch den YeastVisitor über die ComponentManager-Objekte der Bausteine eingestellt.

Für Bausteine, die Unterbausteine besitzen müssen die entsprechenden Modellierungsregeln bestimmt werden. Für jeden Baustein sollte ein eindeutiger Farbwert bestimmt werden, damit Baustein-Objekte im Modell leicht wiedererkannt werden können. Einigen Bausteinen wurde eine Textur zugewiesen.

Um spezielle Richtlinien für einzelne Bausteine formulieren zu können, wurden für einige Bausteine erweiterte ComponentManager-Klassen entwickelt. Manche Bausteine besitzen spezielle Modellierungswerkzeuge im Modellierungstool, die über erweiterte FXComponent-Klassen definiert werden.

Es folgt nun eine Beschreibung aller entwickelten Baustein-Klassen sowie deren spezialisierten ComponentManager- und FXComponent-Klassen.

6.4.1 Die Zelle

Zur Repräsentation einer Zelle im Modell wurde die Klasse *Cell* entwickelt. Ein Objekt dieser Klasse bildet den Basis-Baustein innerhalb der Zellmodellierungsprozesse.

Eine Zelle besitzt einen Zellkern, mehrere Mitochondrien, eine Vakuole, ein Zytoskelett, einen Golgi-Apparat, ein Endoplasmatisches Retikulum und mehrere Vesikel und Zisternen.

Die Zelle ist ein Baustein mit begrenzter Struktur, im Falle einer Hefezelle besteht die Begrenzung aus einer Plasmamembran und einer Zellwand. Für die Darstellung der Zelle wurde eine Textur zugewiesen. Im nächsten Abschnitt werden die implementierten Klassen der Zellbegrenzungen vorgestellt.

6.4.2 Begrenzungen der Zelle

Für die Beschreibung der zwei unterschiedlichen Begrenzungen der Hefezelle, der Plasmamembran und der Zellwand wurden die spezialisierten Border-Klassen *Membrane* und *CellWall* entwickelt.

Die Klasse Membrane wird zusätzlich dazu verwendet, um die Begrenzungen der membranbegrenzten Komponenten zu definieren. Ein Baustein, der die beiden Zellkern-Membranen oder die Membran einer Vakuole beschreibt, ist also ebenso eine Instanz der Membrane-Klasse wie ein Baustein, der die Plasmamembran der Zelle darstellt. Im nächsten Abschnitt werden die für die Hefezelle implementierten Bausteine, die von einem oder mehreren Membran-Objekten begrenzt werden, vorgestellt.

6.4.3 Membranbegrenzte Komponenten

Um einen membranbegrenzten Baustein in das System einzuführen, genügt es in der Regel für den Baustein eine eigene Klasse zu entwickeln und die entsprechenden Begrenzungen über den ComponentManager durch das YeastVisitor-Objekt definieren zu lassen.

Für die begenzten Bausteine werden durch die entwickelten Klassen FXComponent und FXBorder bereits mehrere GUI-Elemente bereitgestellt, durch die die Begrenzungen manipuliert werden können. Es ist im allgemeinen nicht erforderlich, für Border-Bausteine neue GUI-elemente durch erweiterte FXComponent-Klassen zu definieren.

Zur Formulierung von Modellierungsregeln kann bei den membranbegrenzten Bausteinen häufig ebenfalls hinreichend auf die vorhandenen Klassen Component- und BorderManager zurückgegriffen werden, sodass keine spezialisierten Manager-Klassen entwickelt werden müssen.

Maßgeblich verantwortlich für das Aussehen einer membranbegrenzten Komponente ist die äußere Form ihrer Begrenzungen. Über den YeastVisitor werden für die Border-Objekte eines Bausteins Form und Größenintervalle festgelegt.

Membranbegrenzte Bausteine lassen sich sehr einfach in das Modellierungstool einbinden, da bereits viele nützliche Hilfsmittel zur Beschreibung und Verwaltung dieser Objekte vorhanden sind. Für die Hefezellen-Modellierung wurden Bausteine für den Zellkern, Mitochondrien, Vakuolen, Vesikel und Zisternen entwickelt. Um weitere membranbegrenzte Bausteine hinzuzufügen — z. B. einen Plastid-Baustein für die Modellierung von Pflanzenzellen — können die im Folgenden vorgestellten Bausteine als Implementierungsbeispiele herangezogen werden.

6.4.3.1 Zellkern

Die Klasse *Nucleus* beschreibt einen üblichen Zellkern. Die Hefezelle besitzt genau einen Zellkern, der die Form einer Kugel zugewiesen bekommt. Dieser Zellkern besitzt eine Textur, durch die versucht wird, die über einem Zellkern verteilten Kernporen im Modell darzustellen, ohne für diese Komponenten eine eigene Bausteinklasse entwickeln zu müssen (siehe Abb. 6.3). Er enthält genau ein einziges Kernkörperchen (Nucleolus), das eine eigene Baustein-Klasse besitzt, die nun kurz vorgestellt wird.

Nucleolus

Das Kernkörperchen eines Zellkerns wird durch die Klasse *Nucleolus* beschrieben. Der Nucleolus des Zellkerns wird durch eine Kugel mit einer dunklen Textur dargestellt (siehe Abb. 6.4).

Das Aussehen des Kernkörperchens wird durch einen Wert für den Kugelradius definiert. Über die Klasse *NucleolusManager* können Modellierungsrichtlinien für den Radius des Kernkörperchens spezifiziert werden.

Damit die Größe des Nucleolus im Modellierungstool durch den Benutzer bestimmt werden kann, wurden entsprechende GUI-Elemente in der Klasse *FXNucleolus* implementiert, über die der Radius des Kernkörperchens verändert werden kann.

Abbildung 6.3: Ein Zellkern mit Textur aus einem Zellmodell

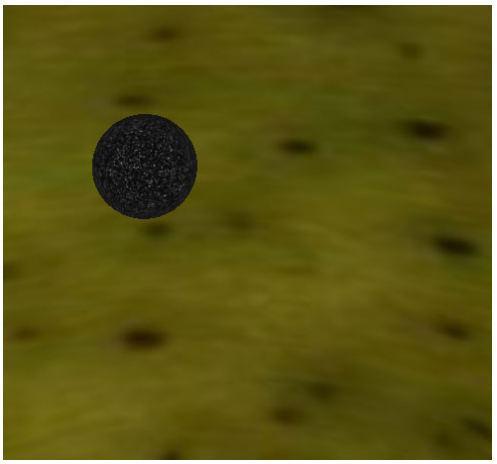

Abbildung 6.4: Innenansicht eines Zellkerns mit Kernkörperchen

6.4.3.2 Mitochondrien

Für die Repräsentation von Mitochondrien im Zellmodell wurde die Klasse *Mitochondrion* entwickelt. Mitochondrien werden von zwei Membranen begrenzt, die innere Membranen eines Mitochondriums besitzt sehr starke Einkerbungen in den Innenraum des Mitochondriums.

Um diese Struktur der inneren Membran darzustellen, könnte eine entsprechende Shape-Klasse entwickelt werden, die eine auf diese Art und Weise gefaltetet Membran parametrisiert. Da für das Mitochondrium allerdings keine Unterbausteine implementiert wurden, die sich innerhalb des membranbegrenzten Bereiches befinden, spielt der innere Aufbau des Mitochondriums bei der Modellierung keine entscheidende Rolle. Der Einfachheit halber wird deshalb zur Modellierung der inneren Membran, wie auch zur Modellierung der äußeren Membran, eine ellipsoidische Shape-Objekt verwendet. Mitochondrien wird eine rötliche Textur zugewiesen (siehe Abb. 6.5).

Über die Anzahl von Mitochondrien in Hefezellen konnten keine genauen Angaben gemacht werden, deswegen können beliebig viele Mitochondrien-Bausteine hinzugefügt werden.

6.4.3.3 Vakuole

Die Klasse *Vacuole* dient zur Beschreibung von Vakuolen. Eine Vakuole besitzt nur eine einzige begrenzende Membran und bekommt für die Modellierung innerhalb der Hefezelle keine Unterbausteine zugewiesen. In der Hefezelle existiert genau eine zentrale Vakuole, deren Form in den Modellierungsrichtlinien als ellipsenförmig definiert wurde.

In der dreidimensionalen grafischen Darstellung wird die Vakuole häufig als trübes, milchiges Objekt gezeichnet. Es wurde versucht die Vakuole durch eine entsprechende Farbe im Modell darzustellen (siehe Abb. 6.5).

6.4.3.4 Vesikel und Zisternen

Als Vesikel werden kugelförmige, als Zisternen ellipsenförmige membranbegrenzte Elemente von Zelle bezeichnet, die nicht die Komplexität von Zellorganellen aufweisen und z. B. zum Transport bzw. zur Speicherung von Stoffen dienen.

Zur Repräsentation von Vesikeln und Zisternen im Zellmodell werden die Klassen *Vesicle* und *Cisternae* bereitgestellt. Vesikel und Zisternen sind die Grundbausteine des Endoplasmatischen Retikulums und der Dictyosomen des Golgi-Apparates, außerdem können sie als intrazelluläre Vesikel und Zisternen für die Modellierung der Zell-Bausteine verwendet werden.

Vesikel und Zisternen besitzen selbst keine Unterbausteine. Die Modellierungsrichtlinien dieser Komponenten werden in Abhängigkeit von dem Baustein formuliert, in dem sie vorkommen. Die Vesikel und Zisternen besitzen im Endoplasmatischen Retikulums, der Dictyosomen und der Zelle jeweils unterschiedliche farbige Darstellungen (siehe Abb. 6.5).

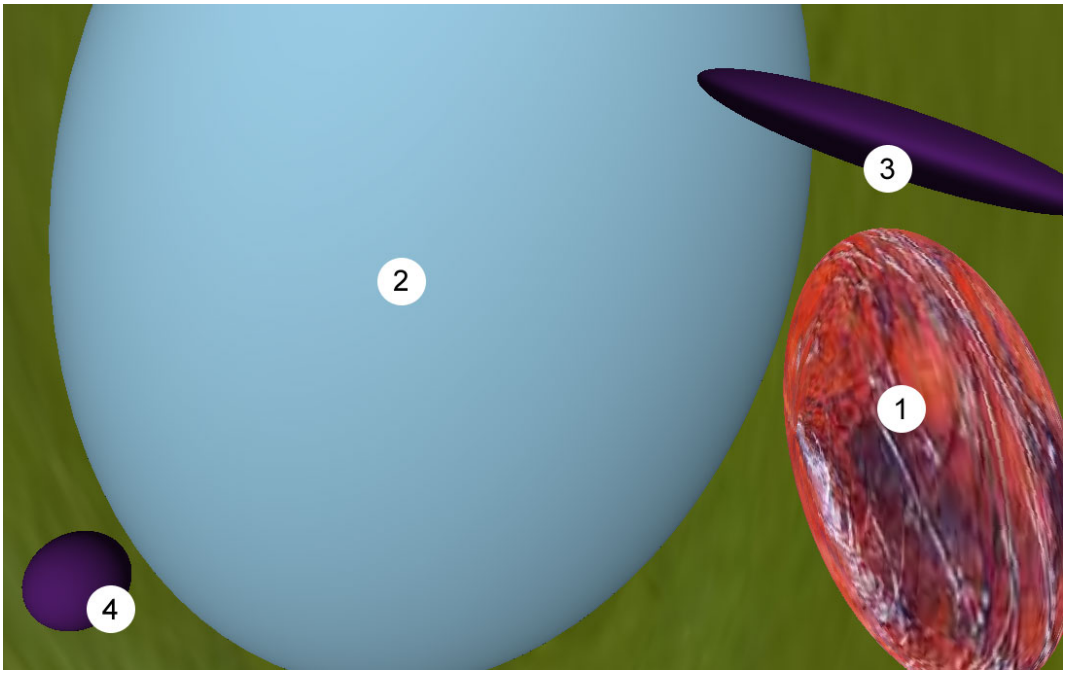

Abbildung 6.5: Bild aus dem Inneren des Zellmodells; *1 - Mitochondrium, 2 - Vakuole, 3 - intrazelluläre Zisterne, 4 - intrazelluläres Vesikel*

6.4.4 Golgi-Apparat

Der Golgi-Apparat einer Zelle wird durch die Baustein-Klasse *GolgiApparatus* repräsentiert. Der Golgi-Apparat wird durch alle Dictyosomen einer Zelle zusammengesetzt und besitzt deswegen eine komplexe Struktur.

Der Golgi-Apparat der Hefezelle besteht aus genau einem Dictyosom-Baustein. Die für die Darstellung der Dictyosomen entwickelten Klassen werden im Folgenden beschrieben.

Dictyosom

Ein Dictyosom besitzt als Baustein-Klasse die Klasse *Dictyosome*. Ein Dictyosom besteht in der Regel aus einem Stapel von 5-8 übereinanderliegenden membranbegrenzten Zisternen. Diese Zisternen können mehrere kugelförmige Vesikel ausbilden.

Vesikel- und Zisternen-Bausteine können zum Modellieren eines Dictyosoms verwendet werden. Um dem Benutzer die Möglichkeit zu geben, vordefinierte Zisternenstapel zu erstellen und ihn somit von dem mühseligen manuellen Aufbau eines Dictyosoms zu befreien, wurde eine Methode entwickelt, durch die es möglich ist, die Darstellung eines Dictyosom zu parametrisieren.

Der Stapel, der durch die Zisternen des Dictyosoms gebildet wird, ähnelt in seiner Form einem Kegelstumpf. Ein solcher Kegelstumpf kann durch die Angabe von folgenden Parametern definiert werden:

- einen Wert für den Radius des Grundkreises
- einen Wert für den Radius des oberen Kreises
- einen Wert für die Höhe des Stumpfes

Wenn zusätzlich zu den Kegelstumpf-Parametern noch die Anzahl der übereinandergeordneten Zisternen, angegeben wird, läßt sich ein Zisternenstapel automatisch generieren (siehe Abb. 6.6). Als Mittelpunkt des Gesamtkomplexes des Dictyosoms wird der Mittelpunkt des beschriebenen Kegelstumpfes übernommen. Für die Klasse

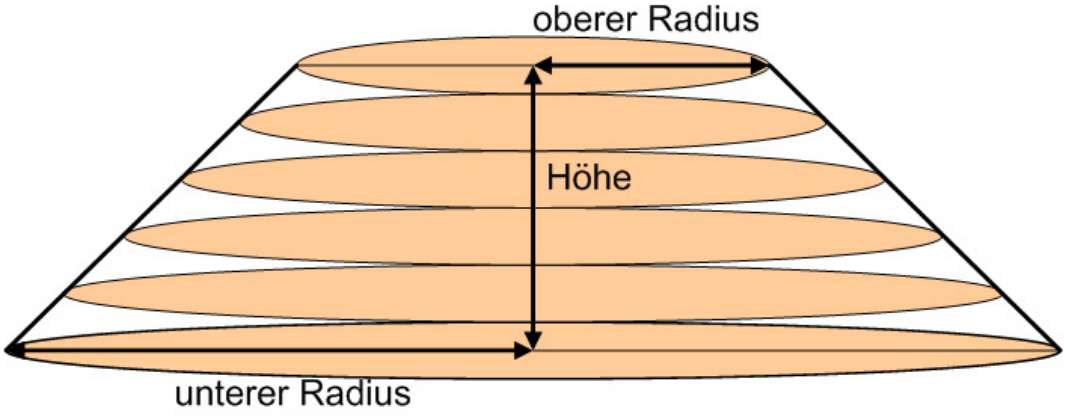

Abbildung 6.6: Darstellung der Parametrisierung des Zisternenstapels eines Dictyosoms

Dictyosom wird eine init-Funktion bereitgestellt, durch die aus den vier Parametern ein neues Dictyosom berechnet wird. Abbildung 6.7 zeigt ein automatisch generiertes Dictyosom in der dreidimensionalen Ausgabe.

Der generierte Aufbau des Dictyosoms kann nach der Erstellung beliebig geändert werden, vorhandene Zisternen können nach Belieben manipuliert oder entfernt, neue Zisternen und auch Vesikel können in das Dictyosom-Modell eingefügt werden.

Für die Dictyosomen wurde eine eigene Manager-Klasse *DictyosomeManager* entwickelt, über die ParameterRestrictions für die Kegelparameter gesetzt werden. Die Modellierungsregeln für die Anzahl der Zisternen des Stapels sind schon durch die aufgestellten ComponentRestrictions definiert.

In der implementierten Klasse *FXDictyosome* wird ein Erstellungsdialog für die Dictyosomen bereitgestellt. Über diesen Dialog können die Parameter für den Kegelstumpf angegeben werden. Die Werte des DictyosomsManagers werden durch den Dialog ausgewertet.

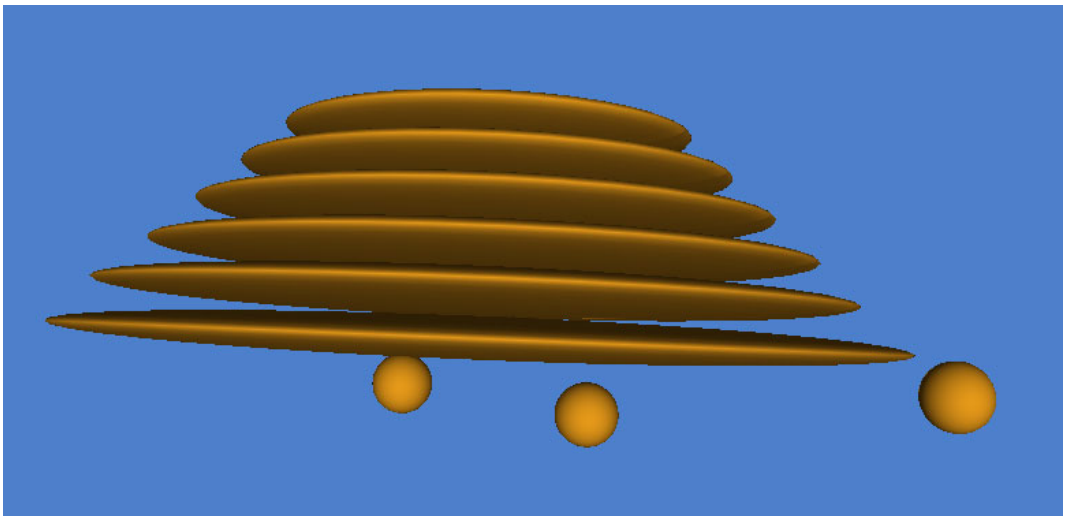

Abbildung 6.7: 3D-Modell eines Dictyosoms

6.4.5 Zytoskelett

Das Zytoskelett der Zelle wird im Zellmodell durch einen Baustein der Klasse *Cytoskeleton* repräsentiert. Ein Cytoskeleton-Baustein besitzt eine komplexe Struktur und dient als Container für alle Zytoskelett-Elemente einer Zelle, wie z. B. die unterschiedlichen Proteinfilamente oder die Zentriolen.

Die Bausteine, die für die Entwicklung des Zytoskeletts der Zelle zur Verfügung gestellt wurden, werden nun in den folgenden Abschnitten vorgestellt.

6.4.5.1 Proteinfilamente

Das Zytoskelett der Zelle wird durch ein Netzwerk von verschiedenen fadenförmigen Proteinfilament-Strukturen gebildet. Zu den Proteinfilamenten zählen z. B. die Intermediärfilamente, die Actinfilamente und die Mikrotubuli.

Wegen der Ähnlichkeit des Aufbaus der Proteinfilamentfäden der unterschiedlichen Filamenttypen wurde eine für alle Proteinfilamente geltende Klasse *Proteinfilament* entwickelt.

Es wird nun ein Ansatz vorgestellt, durch den die Darstellung der einzelnen Filamentfäden parametrisiert werden kann. Ein Filamentfaden kann im einfachsten Fall durch einen länglichen Zylinder dargestellt werden. Zur Parametrisierung des Aussehens eines Filamentfadens können der Radius und die Länge des Zylinders herangezogen werden.

Bei diesem Ansatz können allerdings nur gerade Filamentstücke beschrieben werden, oftmals sind diese Strukturen allerdings kurvenförmig gebogen. Man könnte diese Biegung durch das Zusammenfügen mehrerer entsprechend kleiner Filamentfäden nachbilden, dies könnte allerdings eine enorme Menge an Filamentbausteinen im Modell nach sich ziehen.

Eine weitere Möglichkeit würde darin bestehen, die Filamentfäden nicht nur aus einem, sondern aus mehreren Zylindern zusammenzusetzen. Die kurvenförmige Ausbildung eines Filamentfadens kann modelliert werden, indem die einzelnen Zylinder-Objekte des Filamentfadens gegenseitig verdreht werden.

Diese Art der Darstellung der Proteinfilamente wird von der Klasse Proteinfilament unterstützt, in Abb. 6.8 ist dieser Ansatz nocheinmal bildlich dargestellt. Um eine abgerundete Darstellung an den Übergängen der einzelnen Zylinder zu erzielen, wird in die Verbindungen zweier Zylinder zusätzlich eine Kugel mit gleichem Radius, wie die einzelnen Zylinder einbeschrieben.

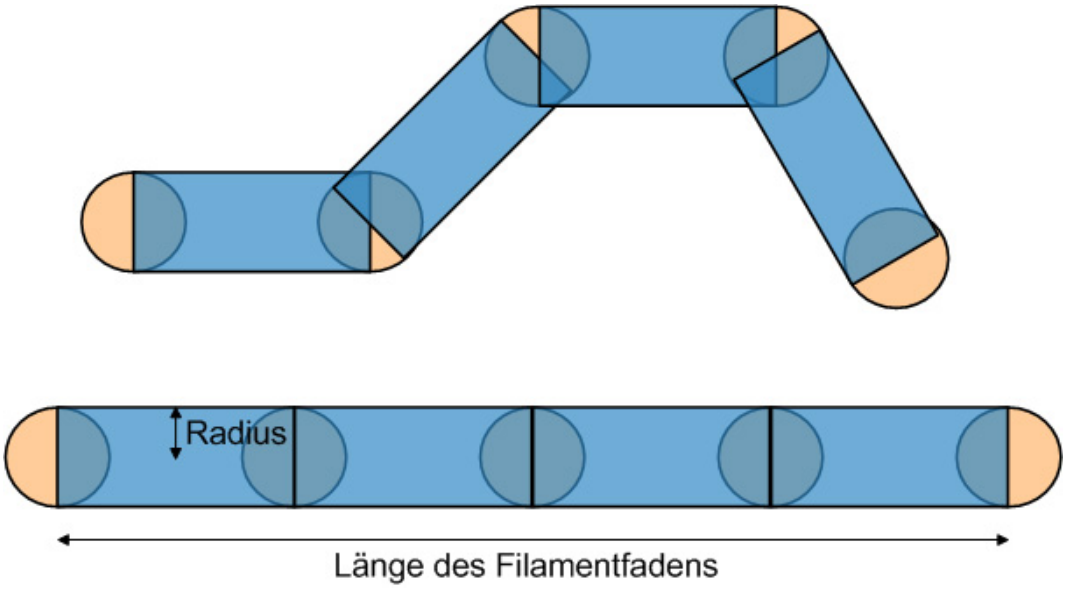

Abbildung 6.8: Darstellung der Parametrisierung eines Proteinfilamentfadens

Die Klasse Proteinfilament stellt eine init-Funktion zur Verfügung, die einen Filamentfaden aus den folgenden Parametern entwirft:

- einem Wert für die Länge des gesamten Fadens
- einem Wert für den Radius der einzelnen Zylinder
- der Anzahl der Zylinder, aus denen der Faden zusammengesetzt wird

Um Modellierungsrichtlinien für die Parameter der Proteinfilament-Bausteine aufstellen zu können, wurde die Klasse *ProteinfilamentManager* entworfen. Zusätzlich werden durch die Klasse *FXProteinfilament* GUI-Elemente bereitgestellt, durch die Proteinfilamentfäden in ihrer Darstellung erzeugt und geändert werden können.

Spezielle Proteinfilamentklassen

Die Klasse Proteinfilaments wird nicht als Baustein-Klasse in der CellLibrary registriert, da sie nur als Zwischenklasse für die unterschiedlichen Filamenttypen verwendet wird und keinen konkreten Baustein beschreibt.

Für jeden Filamenttyp wird eine zusätzliche Klasse in die CellLibrary eingebunden, die zumindest einen eigenen Namen für den Proteinfilamenttyp auszeichnet. Die implementierten Klassen Microtubule, ActinFilament und IntermediateFilament beschreiben die unterschiedlichen Filamenttypen.

Für jeden Filamenttyp werden entsprechende Modellierungsrichtlinien für Länge und Durchmesser aufgestellt, insbesondere wird jedem Filamenttyp ein individueller Farbwert zugeordnet (siehe Abb. 6.9 und Abb. 6.10).

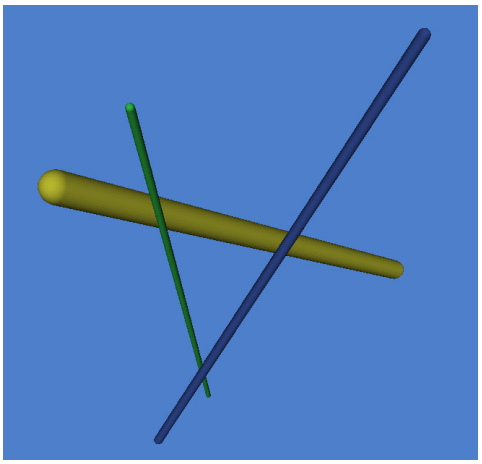

Abbildung 6.9: 3D-Darstellung dreier unterschiedlicher Proteinfilamentfäden; *gelb - Mikrotubuli, blau - Actinfilament, grün - Intermediärfilament*

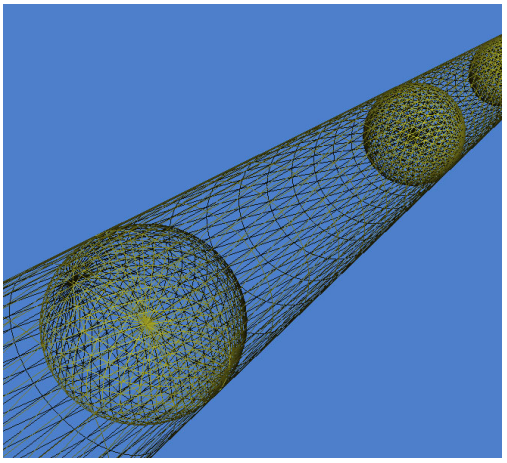

Abbildung 6.10: Drahtgittermodell eines Mikrotubuli

6.4.5.2 Zentriol und Diplosom

In Hefezellen ist in der Regel genau ein Diplosom vorhanden. Diese Komponente wird logisch dem Zytoskelett zugeordnet. Ein Diplosom wird aus zwei Zentriolen aufgebaut, die wiederum aus neun Mikrotubuli-Tripletts zusammengesetzt sind.

Zur Modellierung eines Diplosoms wurden die Klassen *Diplosome*, *Centriole* und *MicrotubuleTriplet* entwickelt. Diese Klassen werden nun kurz vorgestellt.

MikrotubuleTriplet

Ein MicrotubuleTriplet besteht aus drei nebeneinander angeordneter Mikrotubuli-Fragmenten. Für die automatische Generierung der Darstellung von Mikrotubuli-Triplets werden die für die Proteinfilamente aufgestellten Parameter Filamentlänge, Zylinderradius und Zylinderanzahl verwendet.

Da sich die Parametrisierung der Darstellung nicht von der für Proteinfilamente entworfenen Parametrisierung unterscheidet, können für Mikrotubuli-Triplets die Manager-Klasse der Proteinfilamente nutzen, zusätzlich wurde eine neue Klasse FX-MicrotubuleTriplet implementiert.

Centriole

Ein Zentriol wird aus neun Mikrotubuli-Triplets aufgebaut, die einen zylinderförmigen Komplex bilden. Das Aussehen eines Zentriols kann durch Parameter für die Länge und die Höhe des durch die Mikrotubuli-Triplets beschriebenen Zylinders definiert werden.

Für die Centriole-Bausteine wurde eine eigene Manager-Klasse *CetrioleManager* entwickelt. Außerdem werden durch die Klasse *FXCentriole* GUI-Elemente breitgestellt, die zum Erstellen und Verändern von Zentriolen verwendet werden können.

Diplosome

Ein Diplosom besteht aus zwei Zentriolen, die senkrecht zueinander stehen. Zur Parametrisierung der Darstellung werden nur Parameter für die Darstellung der zwei Zentriolen benötigt, die senkrechte Anordnung der Zentriolen kann automatisch erfolgen.

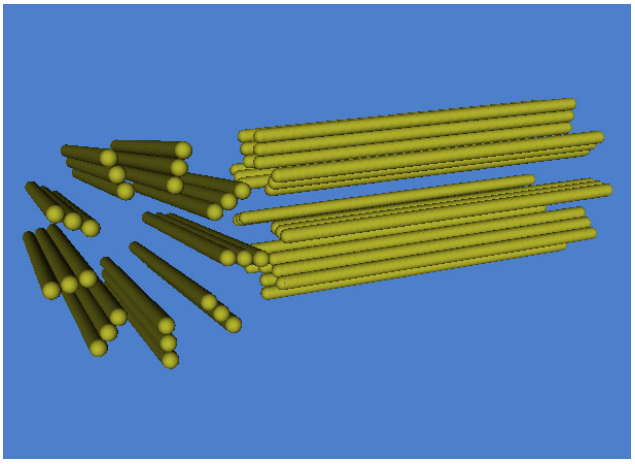

Abbildung 6.11: 3D-Darstellung eines Diplosoms

Diplosome-Bausteine können auf die Manager-Klasse der Centriole-Bausteine zurückgreifen, Windowssystem-spezifische Eigenschaften der Diplosome-Bausteine werden durch die Klasse FXDiplosome beschrieben.

In Abb. 6.11 ist ein 3D-Modell eines Diplosoms dargestellt.

6.4.5.3 Ribosomen

Die Zelle enthält eine Vielzahl von Ribosomen, die entweder an Membranen angelagert sind oder sich frei in der Zelle bewegen. Da ihre Darstellung als eigenständige Einheiten im Zellmodell nicht ausdrücklich vorgesehen ist, wurde für Ribosomen keine Bausteinklasse implementiert.

Unter Umständen können Ribosomen auf den Membranen, auf denen sie vorkommen durch geeignete Texturen angedeutet werden. Das Hinzufügen einer Ribosom-Bausteinklasse ist jederzeit möglich, wenn es die Problemstellungen erfordern sollten.

6.4.6 Endoplasmatisches Retikulum

Baustein-Objekte der Klasse *EndoplasmaticReticulum* stellen das Endoplasmatische Retikulum der Zelle dar. In der Hefezelle ist ausschließlich die granuläre Form des Endoplasmatischen Retikulums zu finden, die entsprechende Baustein-Klasse beschreibt nur diesen speziellen Typ. Um die glatte Form darstellen zu können, muss entweder diese Klasse erweitert oder eine neue Klasse entwickelt werden.

Das Endoplasmatische Retikulum ist eine komplexe Struktur, die aus einer Vielzahl von flachen übereinandergeordneten Zisternen besteht. Darüber hinaus umhüllt es den Zellkern und breitet sich von dort in die Zelle aus.

Zur Modellierung des Endoplasmatischen Retikulums stehen dem Benutzer Zisternen- und Vesikel-Bausteine zur Verfügung. Vesikel sollen dazu dienen, um die Umhüllung des Zellkerns nachzubilden, durch die Zisternen soll die übrige Struktur aufgebaut werden können. Für die Zisternen und Vesikel des Endoplasmatischen Retikulums ist ein eigener Farbwert und eine Textur, durch die die Ribosomen auf den Außenmembranen dargestellt werden, vorgesehen.

7 Evaluierung und Ausblick

In diesem Kapitel werden die bei der Entwicklung des Computermodells der Zelle und des Zellmodellierungstools erzielten Ergebnisse bewertet. Anschließend wird ein Ausblick über mögliche zukünftige Weiterentwicklungen des Zelleditors und der Modellierungsklassen gegeben.

7.1 Evaluierung

Da das entwickelte Programm bislang noch nicht durch unabhängige Personen getestet wurde, basiert die Bewertung der Ergebnisse dieser Arbeit ausschließlich auf eigenen kritischen Betrachtungen. Für die Evaluierung wird zuerst das entwickelte Klassenmodell zur computerbasierten Darstellung einer Zelle und anschließend die implementierte Benutzeroberfläche betrachtet. Am Ende dieses Unterkapitels werden die im vorigen Kapitel beschriebenen konkreten Modellierungsklassen für die Hefezelle bewertet. Abschließend werden noch einige Bemerkungen angefügt, die sich mit der durch das Klassenmodell gegebenen Möglichkeit beschäftigen, andere physische Objekte als biologische Zellen zu modellieren.

7.1.1 Die Basisklassen des Zellmodells

Durch das realisierte Klassenmodell wurde die Problemstellung der Unterstrukturierung der Zelle in einzelne Bausteine und deren Beschreibung durch Funktionen, Attribute und eine dreidimensionale computergrafische Darstellung hinreichend gelöst.

Zusätzlich wurde durch das Einführen von Modellierungsrichtlinien eine Möglichkeit erarbeitet, durch die der Benutzer des Modellierungstools bei seinen Modellierungsprojekten unterstützt werden kann, insbesondere auch durch die Berücksichtigung der unterschiedlichen Zelltypen.

Das Zellmodell und die einzelnen Bausteine lassen sich abspeichern, wodurch eine Weiterverwendung der entwickelten Modelle ermöglicht wird.

Das entwickelte Klassenmodell ermöglicht eine flexible Erweiterung und ist bestmöglich in Hinblick auf zukünftige Aufgaben und Problemstellungen der Zellmodellierung entwickelt worden.

7.1.2 Das Programm CellEdit

Das entwickelte Programm CellEdit bietet die in dieser Diplomarbeit vorgestellte Funktionalität. Es befindet sich bisher allerdings erst in einer sehr frühen Entwicklungsversion und benötigt noch Verbesserungen im Bereich der Bedienbarkeit und muss in Sachen Fehleranfälligkeit und Performance eingehend überprüft werden.

Das Programm ist in der Funktionalität soweit ausgereift, dass es sich automatisch an Veränderungen der Modellierungsklassen anpasst. Es müssen also beim Hinzufügen neuer Zellbausteine, mit Ausnahme der Bereitstellung oder Veränderung von bausteinabhängigen GUI-Elementen, keine Änderungen an der Benutzeroberfläche durchgeführt werden.

In Kapitel 7.2 werden weitere Funktionen vorgestellt, die für die Zellmodellierung von Bedeutung sind, bei dem entwickelten Programm allerdings noch nicht berücksichtigt wurden.

7.1.3 Die Klassen zur Modellierung von Hefezellen

Die in Kapitel 6 beschriebene Implementierung von Klassen zur Modellierung einer Hefezelle stellt ein Beispiel dar, wie einerseits Bausteine für die Zellmodellierung entwickelt werden und andererseits ein zelltypabhängiger Modellierungsbaukasten bereitgestellt werden kann.

Die entwickelten Baustein-Klassen besitzen bislang noch keine semantischen Eigenschaften oder Funktionen und sind nur durch ihr grafisches Aussehen bestimmt. Die Darstellung der Komponenten wird einzig und allein durch drei Elementarobjekte — Kugeln, Ellipsoide und Zylinder — beschrieben, wodurch das Aussehen des modellierten Objektes noch sehr technisch wirkt.

Für das Aufstellen der Richtlinien zur Modellierung einer Hefezelle standen nur wenige Informationen über den Aufbau von Zellen diesen Typs zur Verfügung, wodurch viele Modellierungsregeln nur willkürlich aufgestellt werden konnten.

In Kapitel 7.2 werden notwendige Weiterentwicklungsmaßnahmen beschrieben, durch die realitätsnähere Zellmodelle entwickelt werden können.

7.1.4 Modellierung beliebiger physischer Objekte

Am Ende sei noch auf einen Aspekt hingewiesen, der bei der Entwicklung des Klassenmodells und des Editors zwar keine entscheidende Rolle gespielt hat, aber dennoch für zukünftige Aufgaben von Interesse sein kann.

Die Zelle ist ein physisches Objekt der realen Welt und lässt sich durch Eigenschaften, Verhalten und ein Aussehen beschreiben. Dabei unterscheidet sie sich nicht wesentlich von anderen physischen Objekten der realen Welt. Diese Objekte der realen Welt können häufig auch auf unterschiedliche Art und Weise klassifiziert werden, ähnlich wie dies bei der Klassifizierung der Zelle der Fall ist. Die in dieser Arbeit vorgestellte Vorgehensweise zur Modellierung einer Zelle kann unter Umständen auch auf andere zu modellierende Objekte übertragen werden.

Als naheliegende Erweiterungen, die auch biologische Objekte als Modellierungsgrundlage besitzen, wären Entwicklungen denkbar, die die Modellierung von Viren,

oder von Zellverbänden wie Geweben oder Organen ermöglichen. Darüber hinaus wäre aber auch die Anpassung an nichtbiologische Modellierungsobjekte zu überlegen.

Das entwickelte Klassenmodell wurde sehr allgemein gehalten und nicht ausschließlich in Hinblick auf die Modellierung von Zellen entwickelt, es existieren z. B. die Klassen ComponentLibrary, ComponentVisitor und FXComponentLibrary, die für die Zellmodellierung entsprechend zu den Klassen CellLibrary, CellVisitor und FXCellLibrary erweitert wurden.

7.2 Ausblick

Es wird nun ein Ausblick auf mögliche zukünftige Erweiterungen des implementierten Zellmodellierungs-Fameworks gegeben. Zuerst werden einige interessante Erweiterungsaufgaben vorgestellt, die bei dieser Arbeit aus Zeitgründen noch nicht berücksichtigt werden konnten. Anschließend wird auf die Erweiterung der Modellierungsklassen und der Benutzeroberfläche eingegangen.

7.2.1 Deformieren von Elementarobjekten

Im Unterschied zu technischen Objekten, wie z. B. Autos oder Computer, lassen sich organische Objekte in der dreidimensionalen Computerdarstellung nicht durch analytische oder approximierte Flächen beschreiben, sondern sind sehr unregelmäßig, zufällig geformt.

Um dieses Aussehen zu beschreiben, ist in erster Linie denkbar, diese unregelmäßigen Formen als Elementarobjekte zu beschreiben. Um z. B. die stark verformte innere Membran eines Mitochondriums darzustellen, könnten spezielle Elementarobjekte bereitgestellt werden.

Eine weitere denkbare Methode wäre, ein Werkzeug zur Verfügung zu stellen, durch das es möglicht ist, auf die Flächenmodelle der Elementarobjekte zuzugreifen und diese zu deformieren, indem einzelne Punkte beliebig verschoben werden können. Durch die Triangulierung der Flächenmodelle sind solche Verschiebeoperationen unbedenklich möglich.

Besonders durch die Möglichkeit der Deformation von Border-Bausteinen wäre ein großer Schritt zur Entwicklung realitätsnaher Zellmodelle getan, da ein Großteil der Zelle aus Membranen besteht. Um ein Deformationswerkzeug für Border-Objekte zu starten, könnte ein entsprechender Button in das Werkzeug-Bedienelement der Border-Objekte eingefügt werden.

7.2.2 Kollisionsdetektion

Bislang wird beim Bewegen von Bausteinen innerhalb des Szenenbetrachters noch nicht auf Kollisionen zwischen Bausteinen getestet. Es treten bei der Modellierung zwei unterschiedliche Kollissionsarten auf, das wäre zum einen die Kollission zweier normaler Bausteine und zum anderen die Kollission eines Bausteins mit einer umhüllenden Membran.

Die Kollision zwischen normalen Bausteinen könnte mit Hilfe der Berechnung von Bounding-Körpern und den entsprechenden Testroutinen durchgeführt werden, die

Berechnung von Kollission zwischen Border-Bausteinen und normalen Bausteinen erweist sich als bedeutend schwieriger, da sich der eine Baustein innerhalb des anderen befindet. Besonders wenn man bedenkt, dass durch mögliche Deformationen von Bausteinen diese willkürliche Formen annehmen können, ist eine Kollisionsdetektion auf Basis von Bounding-Körpern sehr ineffizient, womit eine dynamische Kollissionsdetektion nicht zu empfehlen wäre.

Eine weitere Idee wäre, dem Benutzer die Möglichkeit zu geben eine statische Überprüfung von Kollissionen durchführen zu können, durch die die einzelnen Flächen der Elementarobjekte aller Bausteine auf gegenseitige Kollision getestet werden würden. Damit der Benutzer solche Kollissionserkennungsoperationen starten kann, könnte jeder Baustein über das Werkzeug-Bedienfeld einen entsprechenden Button besitzen, der dann nach Betätigen alle Unterelemente des Bausteins auf gegenseitige Kollission testet.

7.2.3 Erweiterung der Modellierungsklassen

Von den implementierten Bausteinen der Hefezelle sollten insbesondere die Bausteine der Proteinfilamente des Zytoskeletts und des Endoplasmatische Retikulums erweitert werden.

Die Proteinfilamente des Zytoskeletts können bisher nur als gerade Stücke initialisiert werden, obwohl ihr innerer Aufbau auch eine mögliche Krümmung vorsieht. Um eine Verformung der Proteinfilamente zu ermöglichen könnte in dem zugehörigen Werkzeug-Bedienfeld ein Button bereit gestellt werden, durch dessen Betätigung ein Werkzeug gestartet wird, durch das der Benutzer die Form der Proteinfilamente verändern kann.

Das Endoplasmatische Retikulum ist sehr schwer darzustellen. Die implementierte Methode sieht vor, dass der Benutzer das Endoplasmatische Retikulum aus mehreren Zisternen- und Vesikel-Bausteinen zusammensetzt. Diese Art der Modellierung ist aufgrund der Ausdehnung und Komplexität des Endoplasmatische Retikulum recht mühsam. Eine Idee wäre, das Endoplasmatische Retikulum automatisch generieren zu lassen, nachdem der Rest der Zelle bereits modelliert wurde. Nach Betätigen des entsprechenden Buttons in der Bausteinleiste könnten ausgehend vom Zellkern mehrere Zisternen schalenförmig um den Kern gelegt werden und sich von dort aus in die Zelle ausbreiten.

Die für die Modellierung einer Hefezelle entwickelten Klassen besitzen bislang noch keinerlei bausteintypische Eigenarten. Die Implementierung von Eigenschaften und Funktionen der Bausteine sollte anhand von den an das Modell gestellten Anforderungen erfolgen.

7.2.4 Erweiterung der Benutzeroberfläche

Mögliche Weiterentwicklungen, die die Bedienung des Programmes verbessern würden, werden nun kurz aufgelistet:

- Das zentrale Element des Zelleditors ist der Szenenbetrachter, der ein perspektivisches Bild der Szene darstellt. Eventuell können Viewports, die orthogonale Ansichten der 3D-Szene zeigen, den Modellierungsprozess erleichtern. Das

Klassen-Konzept des Szenenbetrachters ist so ausgerichtet, dass eine Erweiterung um weitere Viewports mit nicht allzu großem Implementierungsaufwand möglich wäre.

- Die über die einzelnen EventHandler bereitgestellte Maus- und Tastatursteuerung des Szenenbetrachters ist auf intuitive Bedienbarkeit zu überprüfen. Die EventHandler-Objekte können entsprechend einfach verändert bzw. erweitert werden.
- Die Einstellungen des Szenenbetrachters, wie die Ausrichtung des Betrachters oder Hintergrundfarbe, könnten mit dem Zellmodell abgespeichert werden, damit sie beim Beenden eines Modellierungsprojektes nicht verloren gehen.
- Das Programm besitzt bislang noch keine Hilfe-Funktionen. Ein solches Hilfe-System würde die Verwendung des Programms für neue Nutzer vereinfachen und sollte deswegen implementiert werden.
- Die einzelnen Werkzeuge des Zelleditors sind bislang statisch in die Benutzeroberfläche integriert. Es könnte eine Möglichkeit implementiert werden, durch die die GUI-Elemente nach eigenen Wünschen in der Benutzerberfläche angeordnet werden können.
- Als letztes sei die Notwendigkeit der grafischen Gestaltung der Benutzeroberfläche erwähnt, um die Benutzerfreundlichkeit des Programmes zu verbessern.

7.3 Abschließende Bemerkungen

Die dreidimensionalen Darstellung simulierter Zellprozesse wird in Zukunft eine immer größere Rolle spielen, insbesondere wenn man die rasante Weiterentwicklung von Methoden und Werkzeugen zum Erzeugen von Datenbeständen der Zelle auf der einen Seite und die Weiterentwicklung von Simulationsverfahren und die Steigerung der Rechenleistung auf der anderen Seite betrachtet.

Ein praktischer Nutzen im Bereich der Zellsimulation ist für die in dieser Arbeit vorgestellte Zellmodellierungsumgebung bislang allerdings noch nicht in Sicht, dies kann sich aber in den kommenden Jahren ändern. Vielleicht kann eine auf den Ergebnissen dieser Diplomarbeit aufbauende Weiterentwicklung des Modellierungsmoduls eines Tages zur räumlichen Computer-Darstellung biologischer Zellprozesse beitragen.

8 Zusammenfassung

Ein wichtiger Aspekt bei der Simulation von zellbiologischen Prozessen ist die Betrachtung räumlicher Bewegungsprozesse, wie Diffusion oder der Transport von Stoffen. Damit diese Prozesse im Computer dargestellt werden können, muss für die Simulation ein dreidimensionales Modell einer Zelle zur Verfügung stehen. Dieses Modell muss neben den geometrischen Informationen auch das Wissen darüber besitzen, was es darstellen soll. Insbesondere müssen einzelne Zellkomponenten, wie Zellkern oder Mitochondrien, im Modell vorhanden sein, die jede für sich spezielle Funktionen und Eigenschaften besitzen.

In dieser Arbeit wurde eine Methode entwickelt, mit der es möglich ist, ein computergrafisches Zellmodell in einer objektorientierten Datenstruktur abzubilden. Die entwickelte Zell-Repräsentation wurde so konzipiert, dass sie flexibel erweiterbar ist, dass es also möglich ist, die Zelle je nach Anforderungen in Zellkomponenten zu untergliedern und diese entsprechend ihrer semantischen Bedeutung im Modell zu erfassen. Jede einzelne Zellkomponente kann für sich auch wieder in Unterkomponenten untergliedert werden, so dass ein Baukasten-System mit mehreren verschachtelten Modellierungsebenen entwickelt werden kann.

Um eine intuitive Gestaltung eines Zellmodells auf Basis des entwickelten Darstellungsansatzes zu ermöglichen wurde eine Benutzeroberfläche entwickelt, die für diesen Zweck mehrere Modellierungswerkzeuge zur Verfügung stellt. Dieses Framework ermöglicht die Modellierung eines dreidimensionalen Zellmodells nach dem Baukastenprinzip sowie die Veränderung der semantischen Eigenschaften der Zelle und der Zellkomponenten. Das Programm wurde in der Programmiersprache C++ mit Hilfe des FOX Windowing Toolkits und der OpenSceneGraph Szenengraph-API entwickelt.

Das Darstellungskonzept für die biologischen Zellen wurde so allgemein wie möglich formuliert, um den Einsatz für die unterschiedlichsten Problemstellungen zu ermöglichen. Dafür ist eine problembezogene Weiterentwicklung des Konzeptes notwendig. Am Ende der Arbeit wurde das Grundkonzept für die Modellierung von Hefezellen erweitert, was als Beispiel für mögliche Weiterentwicklungen dienen soll.

Die zukünftige Aufgabe ist es nun, Simulationsprogramme zu entwickeln, die die dreidimensionalen Zellmodelle mit den entsprechenden Simulationsmethoden verbinden und somit die räumliche Darstellung simulierter Zellprozesse ermöglichen.

A Die Programm-CD

Das im Laufe der Diplomarbeit entwickelte Programm *CellEdit* wurde auf einer CD der Diplomarbeit beigelegt. Im Verzeichnis */bin* dieser CD befindet sich eine unter Microsoft Windows lauffähige Version des Programms, die ausführbare Datei heißt *CellEdit.exe*.

Im Verzeichnis */CellEdit* sind die Quellcode-Dateien und eine Projektdatei für Microsoft Visual Studio .Net zu finden (*CellEdit.sln*).

B Bilder einer modellierten Zelle

Auf den folgenden Seiten werden Bilder eines 3D-Modells einer Zelle dargestellt, das mit Hilfe des in dieser Diplomarbeit entwickelten Modellierungstools entworfen wurden.

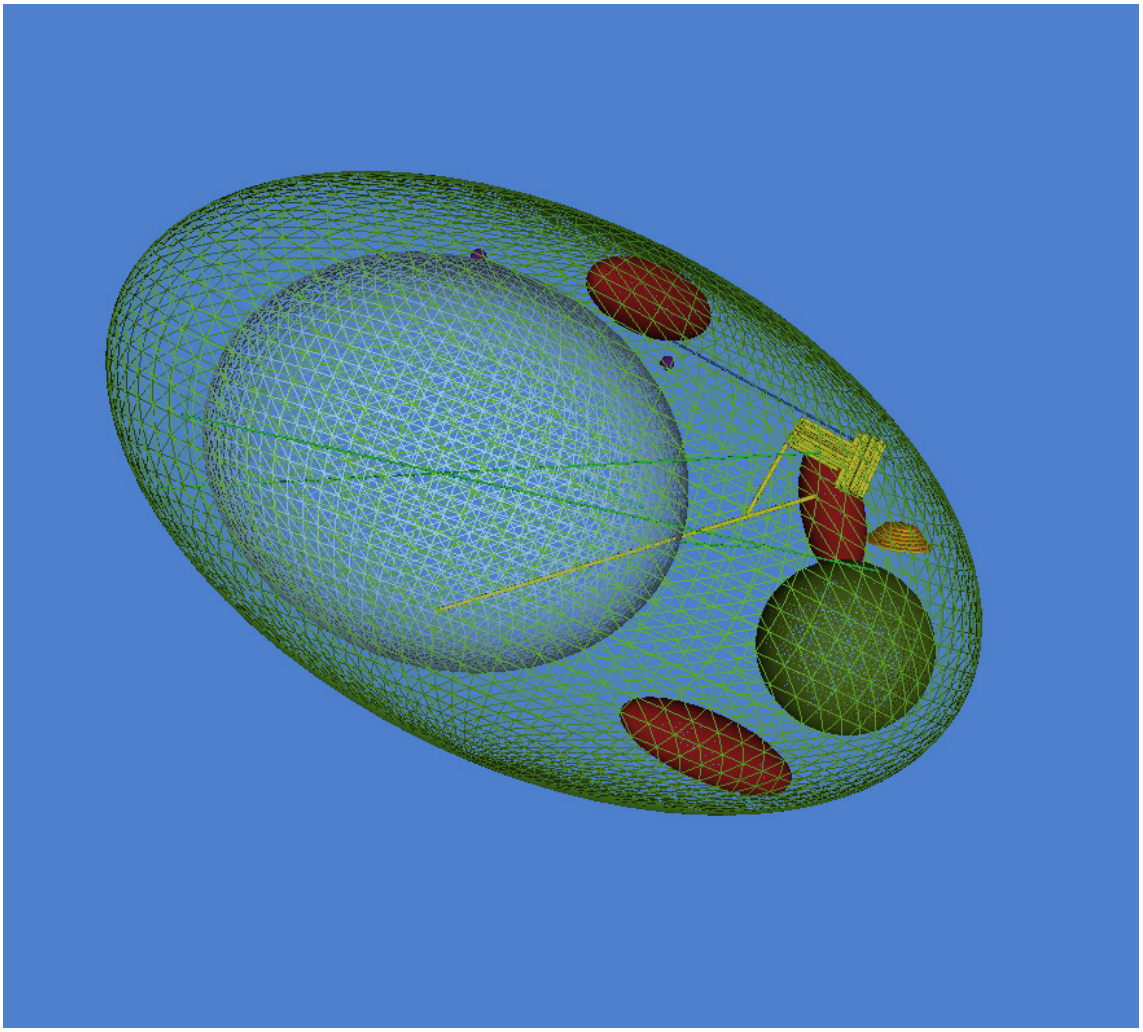

Abbildung B.1: Außenansicht der Zelle (Drahtgittermodell)

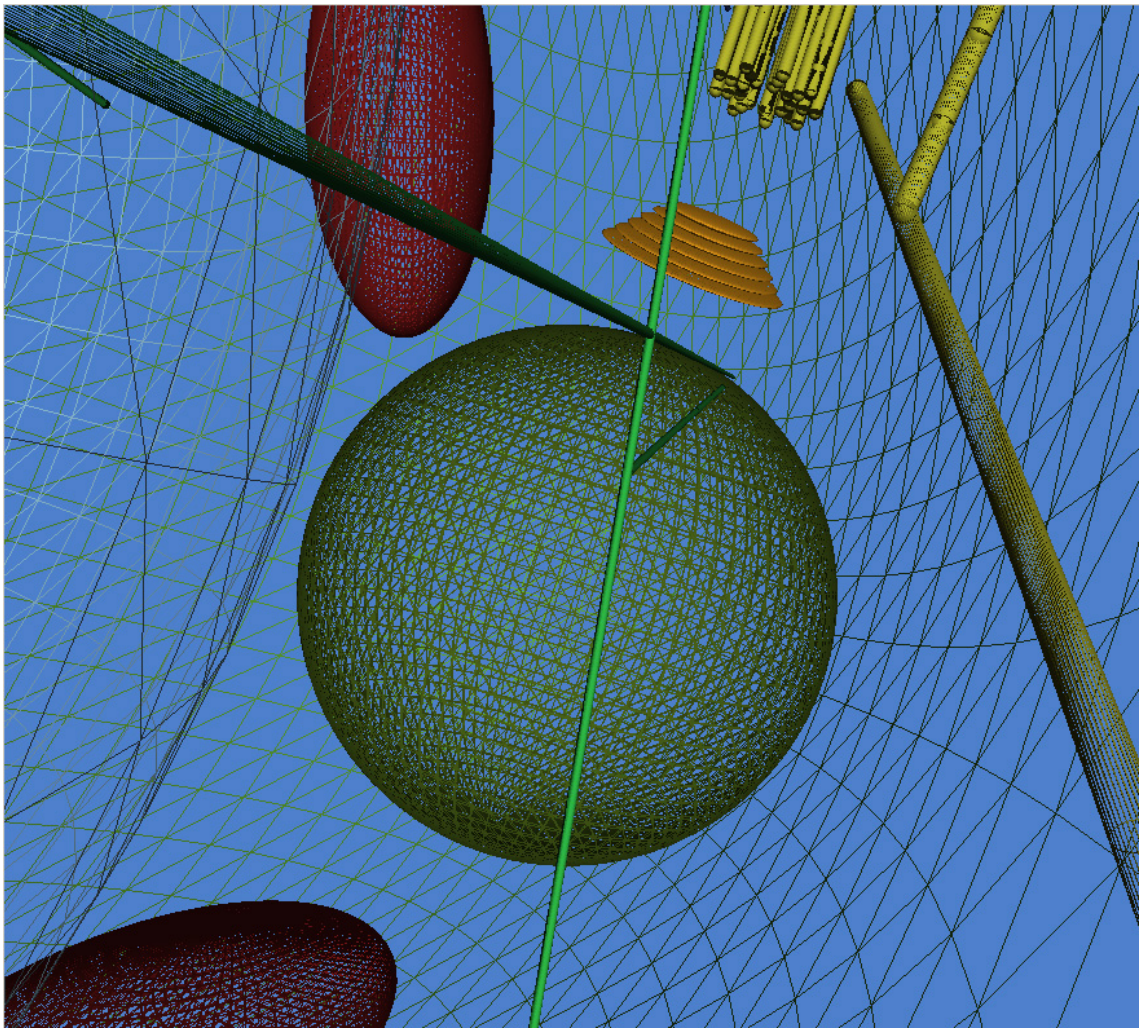

Abbildung B.2: Innenansicht der Zelle (Drahtgittermodell)

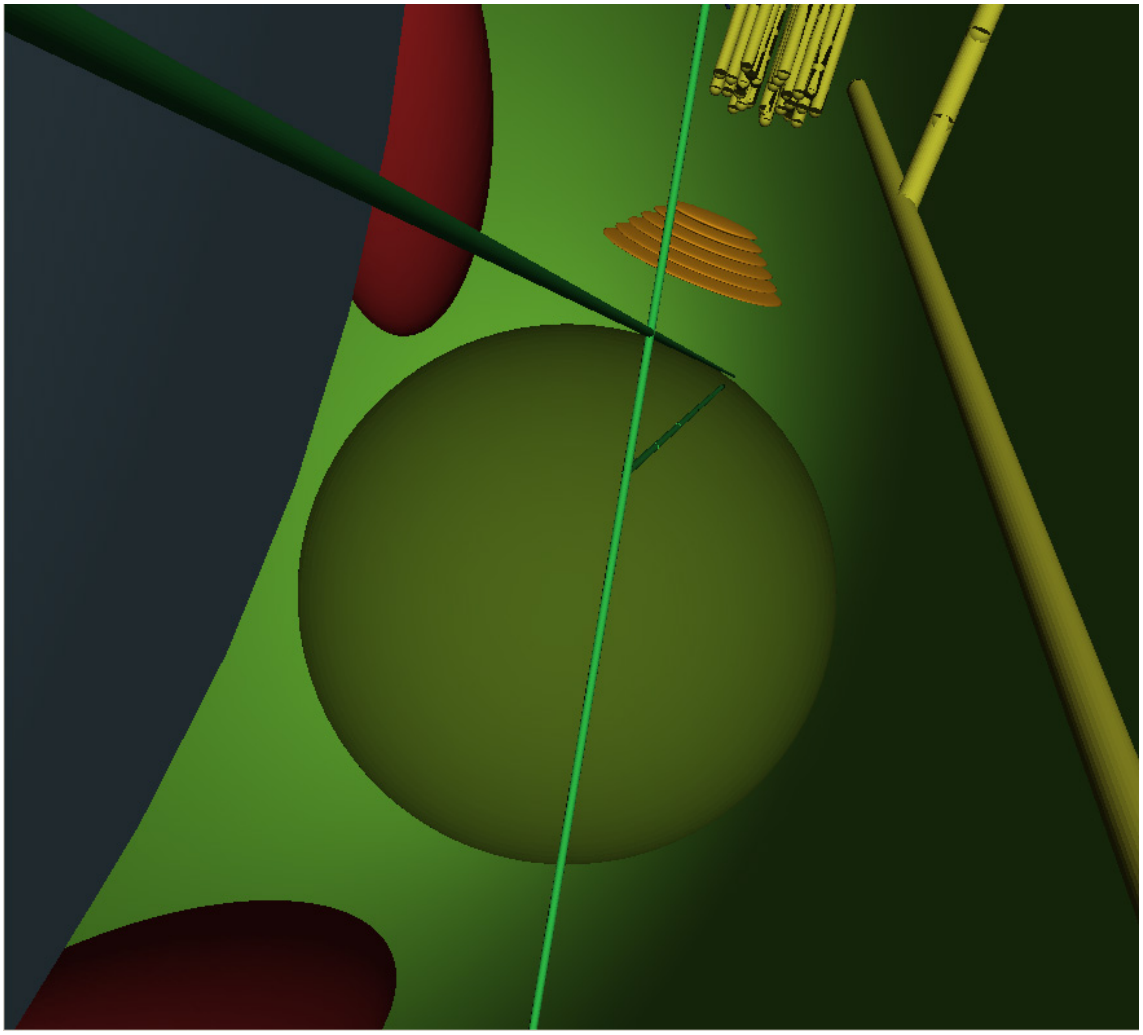

Abbildung B.3: Innenansicht der Zelle (Flächendarstellung)

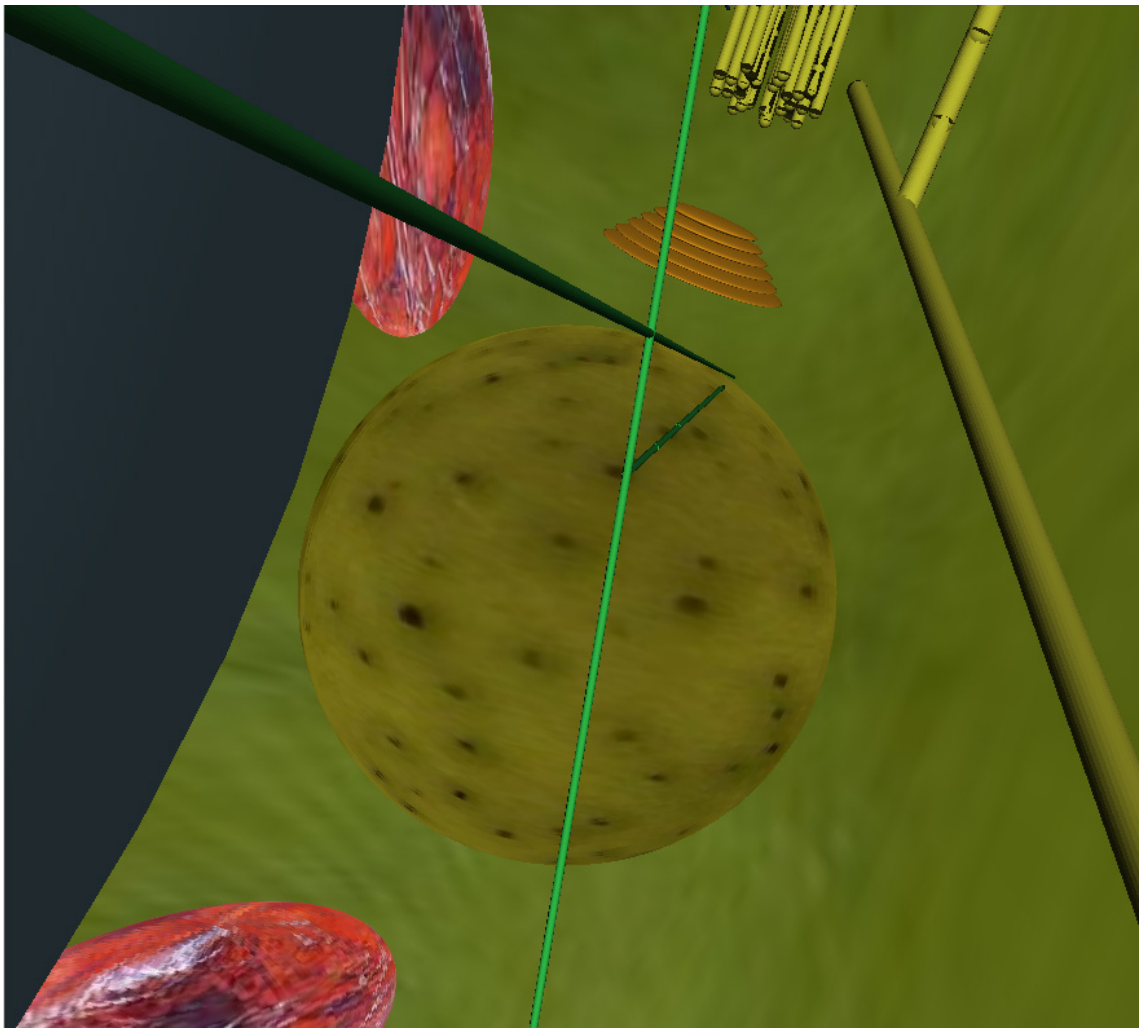

Abbildung B.4: Innenansicht der Zelle (Texturen)

Literatur

[Bart] K. Bart. *Electron Microscopy Images*. http://academics.hamilton.edu/biology/kbart/EMImages.html. 27.11.2003.

[BeGJ01] U. Becker, S. Ganter und C. Just. *Lexikon der Biologie*. Spektrum Akademischer Verlag, Heidelberg, Berlin, Oxford. 2001.

[BevS] A. Bergfeld und Peter v. Sengbusch. *Virtual Plants? - Enhancing Learning with Information Technology*. http://www.biologie.uni-hamburg.de/b-online/snowbird/snowbird/life_cell.htm. 27.11.2003.

[Bio] *Bio-3D*. http://www2.inf.fh-brs.de/~tolry2s/Bio3D. 07.11.2003.

[Biol] *Biokurs 2001*. http://www.merian.fr.bw.schule.de/Beck/skripten/13/bs13-32.htm. 30.11.2003.

[Broc01] *Der Brockhaus multimedial 2002*. Bibliographisches Institut & F. A. Brockhaus AG. 2001.

[ECel] *E-Cell Project*, Institute for Advanced Biosciences. http://www.e-cell.org. 04.11.2003.

[Ende99] C. Ender. *Seminar Computergrafik - Java 3D*, Kapitel 3.1, Scene-Graph Grundlagen. http://www.cg.cs.tu-bs.de/lvcg99/Seminar/java3d.pdf, 1999.

[Faje] M. Fajer. *2010 Biology*. http://fajerpc.magnet.fsu.edu/Education/2010/Lectures/13_Cell_Structure.htm. 27.11.2003.

[Feld01] Horst Feldmann. *Yeast Molecular Biology*, 2001. http://biochemie.web.med.uni-muenchen.de/Yeast_Biology/index.htm. 06.11.2003.

[GHJV98] E. Gamma, R. Helm, R. Johnson und J. Vlissides. *Elements of Reusable Object-Oriented Software*. Addison Wesley. 1998.

[HeSS93] H. Heller, M. Schaefer und K. Schulten. Molecular Dynamics Simulation of a Bilayer of 200 Lipids in the Gel and in the Liquid-Crystal Phases. *Journal of Physical Chemistry 97*, 1993, S. 8343–8360.

[Hitc] 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. 30.11.2003.

[ICel] *A Three-Dimensional Interactive Cell*. http://www.cs.brown.edu/stc/outrea/greenhouse/nursery/biology/home.html. 07.11.2003.

[IdGH01] T. Ideker, T. Galitzki und L. Hood. A New Approach to Decoding Life: Systems Biology. *Annu. Rev. Genomics Hum. Genet. 2001.2*, 2001, S. 343–372.

[JrSw99] Richard S. Wright Jr. und Michael R. Sweet. *OpenGL Super Bible!* Waite Group Press. 2. Auflage, 1999.

[Kita00] H. Kitano. Perspectives in Systems Biology. *New Generation Computing 18*, 2000, S. 199–216.

[KlSi99] H. Kleinig und P. Sitte. *Zellbiologie*. Gustav Fischer Verlag. 1999.

[MaC01] *Mitochondrion and chloroplast*, 2001. http://www.zoobotanica.plus.com/portfolio%20medicine%20pages/organell.htm. 27.11.2003.

[MCDP+96] C.R.F Monks, P.J. Crossno, G. Davidson, C. Pavlakos, A. Kupfer, C. Silva und B. Wylie. Three Dimensional Visualization of Proteins in Cellular Interactions. 1996.

[Nish03] James Nishiura. *Lecture Outline Biology 4 section FV*, 2003. http://academic.brooklyn.cuny.edu/biology/bio4fv/page/cytoskeleton.html. 27.11.2003.

[OnIk00] N. Ono und T. Ikegami. Self-Maintenance and Self-Reproduction in an Abstract Cell Model. *Journal of Theoretical Biology 106*, 2000, S. 243–253.

[Osfia] Robert Osfield. *Introduction to the OpenSceneGraph*. http://openscenegraph.sourceforge.net/introduction/index.html.

[Osfib] Robert Osfield. *OpenSceneGraph*. http://www.openscenegraph.org.

[pVHFr] Dr. phil. Valentina Hinz und Dipl.-Ing. Stefan Franz. *3D - Computergrafiken für Archäologie und Bauforschung*. http://www.hinzundfranz.de/dt/dtmet.htm. 30.11.2003.

[Ross] John Ross. *The Cell*. http://www.jdaross.mcmail.com/mitochon.htm. 27.11.2003.

[Schi] Prof. Dr. R. Schiedermeier. *Blockunterricht WS 1995/96*. http://www.informatik.fh-muenchen.de/~schieder/blockunterricht-ss96/projections.html. 11.12.2003.

[Schi02] Prof. Dr. R. Schiedermeier. *Computergraphik*, 2002. http://www.informatik.fh-muenchen.de/~schieder/graphik-01-02/index.html. 11.12.2003.

[Seif98] Stefan Seifert. *Dreidimensionale Visualisierung des Waldwachstums*, 1998. http://www.wwk.forst.tu-muenchen.de/chair/people/SeifertS/Diplom/Ergebnisse/Vis/BewegungImRaum.html. 30.11.2003.

[SVis] *Seminar Visualisierung und grafische Standards.* http://gdv-serv.prakinf.tu-ilmenau.de/SeminarVis4/. 11.12.2003.

[UdKo02] J. Ude und M. Koch. *Die Zelle - Atlas der Ultrastruktur.* Spektrum Akademischer Verlag. 2002.

[Ulle02] Christian Ullenboom. *Java ist auch eine Insel*, Kapitel 16.1: Das Konzept vom Model-View-Controller. Galileo Computing. 2. Auflage, 2002.

[UML] *The Unified Modeling Language*, Object Managing Group. http://www.uml.org. 04.11.2003.

[VCel] *Virtual Cell Project*, The National Resource for Cell Analysis and Modeling (NRCAM).

[vdZi] Jeroen van der Zijp. *FOX Windowing Toolkit.* http://www.fox-toolkit.org. 04.11.2003.

[VoMü00] Oliver Vornberger und Olaf Müller. *Computergrafik*, 2000. http://www-lehre.informatik.uni-osnabrueck.de/~cg/2002/skript/node113.html. 30.11.2003.

[Wahl98] Günter Wahl. UML kompakt. *OBJEKTspektrum 2*, 1998. http://www.sigs-datacom.de/sd/publications//os/1998/02/OBJEKTspektrum_UM_kompakt.htm. 07.12.2003.

[WoND99] Mason Woo, Jackie Neider und Tom Davis. *OpenGL Programming Guide (Redbook).* Addison Wesley. 2. Auflage, 1999.