TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

M A S T E R A R B E I T

Der Einsatz von Computersimulationssoftware für Nervenzellenstimulation zu Lehrzwecken

ausgeführt am

Institut für Analysis und Scientific Computing

der Technischen Universität Wien

unter Anleitung von

ao. Univ.Prof. Dipl.-Ing. DDDr. Frank RATTAY

durch

Heiko Simon REITNER

A-7535 Güttenbach 340

Juni 2008

Eidesstattliche Erklärung

Ich erkläre ehrenwörtlich, dass ich
die vorliegende Arbeit selbständig
und ohne fremde Hilfe verfasst,
andere als die angegebenen Quellen
nicht benutzt und die den benutzten
Quellen wörtlich oder inhaltlich
entnommenen Stellen als solche
gekennzeichnet habe.

Güttenbach, Juni 2008

Kurzfassung

Mit dem rasanten Anstieg an Rechenleistung und dem hohen Preisverfall bei leistungsfähigen Computern, hat die Computersimulation – die ja im Großen und Ganzen aus der rechenintensiven Lösung von Differentialgleichungen besteht – auch auf handelsüblichen Computern Einzug gehalten. Dies führte auch dazu, dass die Simulation von Nervenzellen einen Aufschwung erlebt hat. Simulationsumgebungen wie NEURON, GENESIS oder NEST ermöglichen heute die brauchbare Simulation von einer einzigen Nervenzelle bis zu ganzen Nervensystemen auf einem einzelnen Computer.

Im Zuge dieser Arbeit kam die Idee, das Simulationstool NEURON auf seine Brauchbarkeit für Unterrichtszwecke zu evaluieren. Es sollte den theoretischen Lehrstoff über Nerven anschaulich illustrieren und das Zusammenspiel der Neuronen demonstrieren. Zu diesem Zweck wurde mit einigen Schülern eines Oberstufengymnasiums sowie mit einigen Studenten der Biologie bzw. Medizin die Simulationsumgebung getestet.

Nach einer eingehenden Evaluationsphase (zuerst allein, anschließend unter fachlicher Anleitung), schilderten die Schüler bzw. Studenten ihre Erfahrungen und Eindrücke mit dem Simulationstool. Gemeinsam wurden anschließend Schwierigkeiten bei der Benutzung aufgezeigt. Die Nutzung des Simulationstools stellte sich gemeinhin als sinnvoll und hilfreich zum besseren Verständnis des Unterrichtsstoffs heraus. Auf Grund dessen wurden gemeinsam mit den Schülern und Studenten, anhand der getätigten Erfahrungen und dabei auftretenden Schwierigkeiten, Verbesserungsvorschläge für die künftige, noch bessere und einfachere Benutzung von NEURON, erarbeitet.

Neben einer allgemeinen biologischen Einführung, findet man in der vorliegenden Arbeit ein umfangreiches Kapitel über NEURON, sowie eine Auflistung vergleichbarer Simulationsumgebungen. Auch der neuen, plattformunabhängigen XML-Modellbeschreibungssprache NeuroML ist ein Kapitel gewidmet. Die ausführliche Schilderung der Evaluierung inkl. Erläuterung der Verbesserungsvorschläge sowie ein Ausblick schließen diese Arbeit ab.

Abstract

Due to the recent significant increase in computing power and the substantial decrease in price of powerful machines (PCs, workstations), computer simulation has experienced rapid growth. Since computer simulation involves mostly the solving of computational-intensive differential equations, the significant increase in computing power now makes it possible to employ simulation software on standard computers. These factors have influenced the simulation of nerve cells and thus have prompted the use of simulation software, such as NEURON, GENESIS or NEST, in the neurosciences. These products, which are executable on normal computers, can simulate situations involving even the minimum of one simple neuron up to complicated nervous systems.

This thesis evaluates the simulation environment NEURON for use at schools or universities, to establish whether simulation software helps students to grasp the theory of nerve cells and nervous systems. In this study, students from a high school and students studying biology and medicine at the university where chosen to test the NEURON simulation tool to determine whether its use is valuable for their learning experience. The students first attempted to use the simulation environment without instruction. Then the students were instructed by an experienced user, who showed them how to use the environment properly. After an exhaustive evaluation period, the students provided feedback on their impressions of and experiences with NEURON. A list was compiled which specified the difficulties and problems which arose while utilizing the software. Despite these obstacles, the students unanimously determined that using a software simulation such as NEURON was valuable and that it helped them to better understand the theory of neuroscience. A list was drawn up containing proposals on which software features could be enhanced or updated to provide a simulation environment better adapted to the user.

In this paper, a general biological introduction is followed by a comprehensive chapter on NEURON. A brief evaluation of other neural simulation environments is included. Furthermore the new platform-independent XML-modeling language NeuroML is described. The paper concludes with a detailed presentation of the student NEURON evaluation, including an extensive exposition of the enhancements suggested by the students.

Danksagung

An erster Stelle möchte ich Herrn Professor Dr. Frank Rattay danken, dass er sich bereit erklärt hat, diese Arbeit zu betreuen und mich dabei zu unterstützen. Er war es, der auf seine begeisternde und unkomplizierte Art in seinen Lehrveranstaltungen zur Computersimulation mein Interesse für diese Thematik geweckt hat.

Weiters sei mein Dank all jenen ausgesprochen, die sich bereit erklärt haben das Tool NEURON zu testen und somit diese Arbeit überhaupt erst ermöglicht haben.

Erwähnen möchte ich an dieser Stelle auch meine Kollegen Dieter und Christian, mit denen ich den Großteil meines Studiums absolvieren durfte. Sie haben maßgeblich dazu beigetragen, dass meine Motivation immer aufrecht blieb, und die gemeinsam verbrachte Zeit – auch außerhalb des Studiums – viel Spaß machte.

Bedanken möchte ich mich auch bei Gerd und Rebecca, die als Korrekturleser fungierten und mich beim Drucken (und darüber hinaus) dieser Arbeit unterstützten.

Ein spezieller Dank geht an meine beste Freundin Martina, die zu aller Zeit für mich da war. Weiters danke ich auch all jenen Freunden, die mir während der Zeit der Erstellung dieser Arbeit zugehört und mich immer wieder aufs Neue motiviert haben.

Schließlich möchte ich noch meinen Eltern Rudolf und Imelda danken, die mir schon sehr früh die Möglichkeit und Freiheit gegeben haben, meinen eigenen Weg zu gehen. Ohne ihr Verständnis und ihre Unterstützung wären mein Studium und meine Entwicklung in dieser Form nicht möglich gewesen. Außerdem danke ich noch meinen Großeltern und meinen Paten, denen mein Wohlergehen immer ein großes Anliegen war.

Vergelt's Gott!

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Als sich gegen Ende meines Masterstudiums der Medizinischen Informatik an der TU Wien die Frage auftat, in welchem Bereich und zu welchem Thema ich meine Masterarbeit verfassen sollte, war die Antwort nicht lange zu suchen. Bereits während des Bakkalaureatsstudiums der Medizinischen Informatik kam ich zum ersten Mal in Kontakt mit der Computersimulation im Rahmen einer einführenden Lehrveranstaltung von Prof. Dr. Frank Rattay [1], [14]. In dieser Lehrveranstaltung wurden verschiedene Computersimulationsprogramme vorgeführt, und es wurde auf allgemeine Aspekte der Computersimulation eingegangen. Es wurde aber auch schon ein spezielles Augenmerk auf die Simulation von Nervenzellen gelegt. Dies weckte mein Interesse an der Computersimulation, da es sich ja bestens in mein eigentliches Fachgebiet - die Medizinische Informatik – einfügte.

Um mich weiter mit der Thematik vertraut zu machen, besuchte ich die fortführenden Lehrveranstaltungen bei Prof. Rattay. In der Lehrveranstaltung „Nervenmodelle“ [2] wurden die Grundlagen und Funktionsweisen von Nervenzellen erläutert. Dies war für mich Neuland aufgrund des nicht vorhandenen Biologieunterrichts während meiner Oberstufenschulzeit (HTBL). In der LVA „Hörtheorie“ wurden verschiedene Studien und Funktionsweisen des Hörapparates illustriert. Es wurde vor allem auf aktuelle Forschungsstandpunkte von Cochlea-Implantaten eingegangen. Schließlich sei noch die LVA „Brain Modelling“ zu erwähnen, die als Ringvorlesung gehalten wird und sich fortwährend mit aktuellen Bereichen der Simulation verschiedenster Nervenzellen auseinandersetzt.

Nach dem Besuch der genannten Lehrveranstaltungen hatte ich mir einen ausreichenden Überblick über die Computersimulation im Allgemeinen, vor allem aber über die Computersimulation im medizinischen Bereich angeeignet. Das geweckte Interesse an der Thematik motivierte mich auch zwei Sommerkurse in diesem Bereich zu besuchen. Es waren dies ein Kurs in Tampere, Finnland, zum Thema „Bioelectrical Signals of the Human Body“ sowie ein Kurs in Lund, Schweden, zum The-

ma „Sim Robots – Out in Space now“. Somit war es für mich klar, dass ich meine Masterarbeit in diesem Bereich verfassen möchte.

Bei der Suche nach einem konkreten Thema wollte ich aber auch mein zweites Standbein, das Informatikmanagement, einfließen lassen. Dieses Masterstudium beschäftigt sich neben einer Auswahl verschiedenster Themen aus dem Bereich der Informatik vorrangig mit Themen der Pädagogik und dem Einsatz der Informationstechnologie im Unterricht.

Insofern war es naheliegend einen Bereich zu finden, der sich beiden Aspekten widmet: einerseits der Computersimulation von Nervenzellen und andererseits dem möglichen Einsatz dieser Simulationsumgebungen in der Lehre im Allgemeinen. Dies brachte mich schließlich zum Thema dieser Masterarbeit: **„Der Einsatz von Computersimulationssoftware für Nervenzellenstimulation zu Lehrzwecken.“**
Da dieses Gebiet noch wenig erforscht bzw. dokumentiert ist, bot es sich sprichwörtlich an, zu diesem Thema eine Arbeit zu verfassen. Meine eben beschriebenen interdisziplinären Interessen und Vorkenntnisse wirken sich auf diese Wahl sicherlich auch positiv aus.

Die konkreten Fragestellungen und Überlegungen zu diesem Thema werden im kommenden Kapitel genauer erörtert. Dort findet man auch die Gliederung und den Aufbau dieser Masterarbeit.

1.2. Themenbeschreibung – Problemdefinition

Die Idee dieser Masterarbeit ist es, die aktuellen Möglichkeiten im Bereich der Simulation von Nervenzellen zu analysieren und festzustellen. Da die Rechenleistung von Haushaltscomputern in den letzten Jahren rapide angestiegen ist und bereits ziemlich umfangreiche Simulationen auf einem handelsüblichen Rechner berechnet werden können, hat sich natürlich auch der Bereich Computersimulation maßgeblich weiterentwickelt. Während man vor noch gar nicht allzu langer Zeit für eine einfache Neuronensimulation ein Netzwerk aus Großrechnern brauchte, kann man heutzutage

komplizierte Neuronennetzwerke auf einem einfachen PC oder MAC simulieren. Erst diese Entwicklung ermöglichte es, die vernünftige Computersimulation der breiten Masse zugänglich zu machen. Bisher war sie ja hauptsächlich dem universitären Bereich und der Forschung vorbehalten.

Nach der eingehenden Untersuchung des Status quo der Computersimulation im Bereich der Nervenzellenstimulation sollen die Möglichkeiten des Einsatzes zu Lehrzwecken eruiert werden. Es gibt einige freie Programme zur Simulation und auch Rechenleistung ist genügend vorhanden. Also zumindest von der technischen Seite sollte es hier keine Hindernisse geben. Wie sinnvoll der Einsatz dieser Simulationstools jedoch ist, soll Hauptaugenmerk dieser Masterarbeit werden. Themen, die diesbezüglich abgehandelt werden, sind etwa, ob ein Einsatz bereits im Gymnasialunterricht denkbar und nützlich sein könnte. Dazu müssten die Simulationsprogramme einfach verständlich und über eine graphische Oberfläche zu bedienen sein. Sie könnten unterstützend sowie zum besseren Verständnis der Funktionsweise eines Nervensystems eingesetzt werden (z. B. im Biologieunterricht). Auch ein Einsatz in der universitären Lehre (z. B. in der Medizin oder in den allgemeinen Naturwissenschaften) könnte denkbar sein. Da die Simulationstools oft nur kompliziert und sperrig in der Bedienung sind und zudem meist Programmierkenntnisse voraussetzen, sollen die aktuellen Simulationsprogramme diesbezüglich getestet und auf ihre Brauchbarkeit im Unterricht analysiert werden.

Im Speziellen soll hier das Nervenzellensimulationsprogramm NEURON [3] evaluiert werden. Es besitzt eine graphische Oberfläche und ist bereits relativ weit verbreitet. Es wird weltweit in den verschiedensten Bereichen eingesetzt und hat bereits einen gewissen Bekanntheitsgrad erreicht. Überdies gibt es große öffentliche Datenbanken [5], in denen Neuronenmodelle abgelegt sind, die von jedermann verwendet werden können. Dies ist ideal für schulische oder universitäre Veranschaulichungszwecke, da man auf bereits vorgefertigte Modelle zugreifen kann und diese nicht aufwändig selbst erstellen muss.

Weiters sollen auch andere Simulationsprogramme, die auf Nervenmodelle spezialisiert sind, evaluiert werden, um einen Vergleich anstellen zu können. Neben

NEURON ist hier das Simulationstool GENESIS gemeint. Natürlich sollen auch andere Programme verglichen und evaluiert werden. Vor allem in der Bedienung und Handhabung sowie im Leistungsumfang gibt es hier mitunter gravierende Unterschiede. Überdies gibt es auch allgemeine Computersimulationsprogramme, wie zum Beispiel MATLAB. Auch diese Tools sollen auf ihre Tauglichkeit in der Neuronensimulation geprüft werden, natürlich mit einem speziellen Augenmerk auf den Einsatz zu Unterrichts-/Lehrzwecken.

Ein weiteres Problem der verschiedenen Computersimulationsprogramme ist deren Inkompatibilität untereinander. Ein in NEURON erstelltes Modell kann nicht einfach exportiert werden und zum Beispiel mit GENESIS weiterberechnet werden. Da die meisten dieser Simulationsumgebungen im universitären Umfeld entstanden sind und ursprünglich für den Eigengebrauch gedacht waren, hatte man nie an einheitliche Modellbeschreibungsformen gedacht. Auch bezüglich Erweiterungsmöglichkeiten sind diese Tools ziemlich eingeschränkt. Mit der Etablierung des XML-Standards [8] zur allgemeinen Beschreibungssprache hat sich auch im Bereich der Neuronensimulation eine allgemeine Neuronenbeschreibungssprache entwickelt. Dieser XML-Standard wird NeuroML [6] genannt und soll in dieser Masterarbeit ebenfalls betrachtet werden. Von der Idee her wäre es sehr sinnvoll, wenn man ein Modell zuerst allgemein beschreiben und anschließend mit verschiedenen Simulationsumgebungen berechnen könnte. Welche Anforderungen an einen allgemeinen Beschreibungsstandard gestellt werden und welche Vorteile und Nachteile ein solcher XML-Standard mit sich zieht, soll genauer erörtert werden. Auch hier soll ein aktueller Ist-Zustand erhoben werden, inwieweit der Standard schon entwickelt und verwirklicht ist und inwieweit die aktuellen Simulationsumgebungen den XML-Standard bereits unterstützen. Eine gut funktionierende Unterstützung eines allgemeinen und lesbaren Standards wäre nicht nur für Forschungs- und Entwicklungszwecke von Vorteil sondern auch für das Kernthema dieser Arbeit: für Lehrzwecke.

1.3. Struktur und Aufbau der Arbeit

Am Anfang der Arbeit soll es eine allgemeine Einführung in die Funktionsweise von Nervenzellen und Nervensystemen geben. Hier soll aus biologischer Sicht der Aufbau einer Nervenzelle samt seinen Zugehörigkeiten illustriert werden. Ohne einem eingehenden Verständnis der biologischen Abläufe und Mechanismen eines Nervensystems ist ein Übergang zur Modellbildung und Computersimulation zwecklos. Die Funktionsweise der Ionenkanäle ist ja für die Simulation maßgeblich. In diesem Kapitel sollen auch die Grundlagen der Modellbildung und deren geschichtliche Entstehung behandelt werden. Speziell zu erwähnen ist hier auch das Hodgkin-Huxley-Modell [7], weil es sozusagen das „Urnervenmodell“ darstellt.

An das einführende Kapitel fügt sich das Hauptkapitel, in dem die Computersimulationsumgebung NEURON eingehend beschrieben wird, an. Hier werden von der geschichtlichen Entstehung bis zum Einsatz der Simulationsumgebung alle relevanten Aspekte abgehandelt. Das Kapitel ist deswegen so umfangreich, damit man sich einen generellen Überblick über eine aktuelle Simulationsumgebung verschaffen kann. Das Programm wird anhand eines Beispiels auf seine Möglichkeiten hin ausgelotet. Damit kann man einen ersten Einblick in die vielen Aspekte der Simulation von Nervenzellenstimulationen erhalten. Auch die internen Abläufe und Berechnungs-/Integrationsmethoden sollen hier erörtert werden.

Anknüpfend an dieses Kapitel folgt die Evaluierung von vergleichbaren Computersimulationsumgebungen. Es soll kurz beschrieben werden, welches Tool für welche Teilbereiche der Simulation gedacht ist und wo man sie am besten einsetzen könnte. Wie schon eingangs erwähnt, sollen hier auch allgemeine Simulatoren auf ihre Tauglichkeit in der Neuronenmodellierung geprüft werden.

Schließlich fügt sich hier das Kapitel zur allgemeinen Beschreibung von Nervenmodellen ein: NeuroML. Zuerst werden allgemeine Anforderungen an einen gemeinsamen Modellierungsstandard aufgelistet und schließlich die Möglichkeiten sowie der Ist-Zustand des XML-Standards NeuroML evaluiert.

Im abschließenden Kapitel werden dann die Möglichkeiten des Einsatzes von Computersimulationsprogrammen zu Lehrzwecken in der Neuronenforschung sowie für Veranschaulichungs- und Verständniszwecke diskutiert. Die Simulationsumgebung NEURON wird von Gymnasialschülern sowie von Studenten der Biologie und Medizin (ohne Programmierkenntnisse) getestet. Basierend auf deren Eindrücken und Erzählungen sollen der Ist-Zustand festgestellt und eventuelle Verbesserungsmöglichkeiten aufgezeigt werden. Im anschließenden Ausblick sollen alle bisher erörterten Aspekte zusammengeführt werden und über die künftigen Möglichkeiten der Nervenzellensimulation, vor allem im Bereich der Lehre und Forschung, nachgedacht werden.

2. Das Nervensystem

2.1. Biologische Grundlagen

Bevor wir in die Welt der Nervenzellenmodellierung eintauchen, ist es wichtig die Struktur und den Aufbau einer Nervenzelle [9] zu verstehen. Überdies ist es notwendig zu wissen, wie eine Nervenzelle arbeitet und wie das gesamte Nervensystem interagiert. Dieses einführende Kapitel soll uns einen kurzen Überblick über alle wichtigen Termini geben und ein besseres Verständnis des Nervensystems schaffen, um die spätere Modellierung und Stimulation der Nervenzellen zu ermöglichen [15].

Die Gesamtheit des Nervengewebes, welches selbst aus Milliarden von Nervenzellen besteht, wird als Nervensystem [16] bezeichnet. Die einzelne Nervenzelle, welche auch Neuron genannt wird, ist mit anderen Neuronen in einer komplexen Anordnung verbunden. Diese Neuronenverbünde haben die Eigenschaft, mittels elektrochemischer Signale eine Vielzahl von Stimuli weiterzuleiten: innerhalb des Nervengewebes als auch von und zu vielerlei anderer Gewebsarten. Im Gehirn allein arbeiten in etwa 10^{10} Neuronen. Seine Aufgabe ist es, Informationen zu erfassen, zu speichern, zu verarbeiten und weiterzuleiten. Generell gliedert man das Nervensystem in zwei Teile, das Zentrale Nervensystem (ZNS) und das Periphere Nervensystem (PNS). Das ZNS, das den weitaus größeren Teil darstellt, besteht aus Gehirn und Rückenmark. Die Spinalnerven (welche dem Rückenmark entspringen) sowie alle außerhalb des ZNS liegenden Nervenzellen gehören zum PNS. Ein direkter Zugriff auf das ZNS von außen ist nicht möglich. Reize von außen, egal ob positiver (z. B.: angenehme Berührungen) oder negativer (z. B.: qualvolle Schläge) Natur, werden erst vom peripheren zum zentralen Nervensystem transportiert. Schmerz etwa wird erst nach einer Verarbeitung durch das Gehirn als solcher empfunden. Angenommen man fügt einer Körperstelle Schmerzen zu und durchtrennt im Vorhinein die Nervenzellen dieser Stelle im Gehirn, so kann man keinen Schmerz fühlen. Natürlich sind auch alle anderen Empfindungen davon betroffen.

Zusätzlich kann man das Nervensystem auch auf einer weniger anatomischen Ebene aufteilen, nämlich auf einer funktionellen Ebene. Dies geschieht in Abhängigkeit von

der Rolle die von den verschiedenen Reizleitung übernommen wird und unabhängig davon, ob sie dem ZNS oder PNS angehören. Man spricht hier vom somatischen und vegetativen Nervensystem. Das somatische Nervensystem ist unmittelbar vom Willen der Person beeinflussbar, wie z. B. die Bewegung von Muskeln. Dem gegenüber gestellt ist das vegetative Nervensystem, welches nur sehr schwer durch den eigenen Willen beeinflussbar ist. Es regelt und kontrolliert unter anderem die inneren Organe.

Schließlich kann man Neuronen noch nach den Richtungen unterteilen, in welche sie Impulse übertragen. Afferente Neuronen übermitteln Impulse vom Rezeptor zum ZNS respektive Gehirn (z. B. Schmerz) und efferente Neuronen übertragen Impulse. Die einzelnen Impulse bezeichnet man nun als Aktionspotentiale [13]. Obwohl Aktionspotentiale von vielen verschiedenen Zellen gebildet werden können, werden sie vorrangig von Nervenzellen zu Kommunikationszwecken genutzt. Die Anzahl der Aktionspotentiale hängt unter anderem von der Länge und Intensität der Stimulation ab.

Neuronen müssen geschützt und versorgt werden. Diese Aufgabe wird durch die sogenannten Glia- oder Stützzellen bewerkstelligt. Sie sind nicht-neuronale Zellen und haben vier Hauptaufgaben:

- sie umgeben die Neuronen und fixieren sie in ihrer Position
- sie versorgen die Neuronen mit Sauerstoff und Nahrung
- sie isolieren Neuronen voneinander
- sie entsorgen abgestorbene Neuronen

Früher nahm man an, dass Gliazellen [17] keine chemischen Synapsen haben und dass sie keine Neurotransmitter [18] freigeben. Heute wissen wir, dass Gliazellen die Reparatur von Nervenzellen regulieren. Obwohl sie im ZNS die Reparatur verhindern, begünstigen sie im PNS diese aber. Verantwortlich dafür sind die sogenannten Schwann'schen Zellen. Sie sind eine spezielle Variante der Gliazellen, die eine Myelinisolierung für periphere Nervenzellen zur Verfügung stellen. Erwähnenswert sind sie, weil sie die Leitfähigkeit der Neuronen verbessern. Die Myelinhülle der

Schwann'schen Zellen verringert die Membrankapazität im Axon und erhöht deshalb die Impulsausbreitungsgeschwindigkeit.

2.2. Die Nervenzelle

Die Nervenzelle, auch Neuron genannt, besteht im Wesentlichen aus dem Zellkörper (=Soma) und den Zellfortsätzen. Auf Grund der verschiedenen Funktionen die von den Nervenzellen in den verschiedenen Teilen des Nervensystems ausgeführt werden, treten die Neuronen in diversen verschiedenen Formen, Größen und elektrochemischen Eigenschaften auf. Der Zellkörper besteht aus zwei wesentlichen Teilen, nämlich dem Zellkern (=Nukleus) und dem Zytoplasma mit den Zellorganellen. Hier passiert der Großteil der Proteinsynthese, und auch der Zellstoffwechsel findet hier statt. Die zwei dafür hauptverantwortlichen Zellorganellen sind das endoplasmatische Retikulum sowie die Mitochondrien, wobei sich das endoplasmatische Retikulum für die Proteinsynthese verantwortlich zeichnet und die Mitochondrien für die Energieproduktion zuständig sind. Der Durchmesser eines Somas bewegt sich in einer Spanne von 4 bis 100 μm und der Durchmesser des Zellkerns beträgt etwa 3 bis 18 μm.

Der Zellkörper ist essentiell für das Überleben der Zellfortsätze. Hätten die Zellfortsätze keine Verbindung zum Zellkörper, würden sie absterben. Man unterteilt die Fortsätze in Dendriten [10] und Axone [11]. Die Dendriten (aus dem griechischen: der Baum) sind oft 5000 oder mehr Ausstülpungen des Zytoplasmas, und sie übertragen elektrische Impulse von anderen Zellen zum Zellkörper. Elektrische Stimuli kommen über die sogenannten Synapsen zu den Dendriten. Diese sind über den gesamten Dendritenbaum verteilt. Die Dendriten summieren die synaptischen Eingänge und sind verantwortlich dafür, ob ein Aktionspotential vom Zellkern produziert wird. Zwar gibt es die Möglichkeit, dass eine Potentialübertragung von den Dendriten, z. B. direkt zu anderen Neuronen, stattfindet, jedoch kann dies nicht durch die chemischen Synapsen passieren, weil diese unidirektional sind.

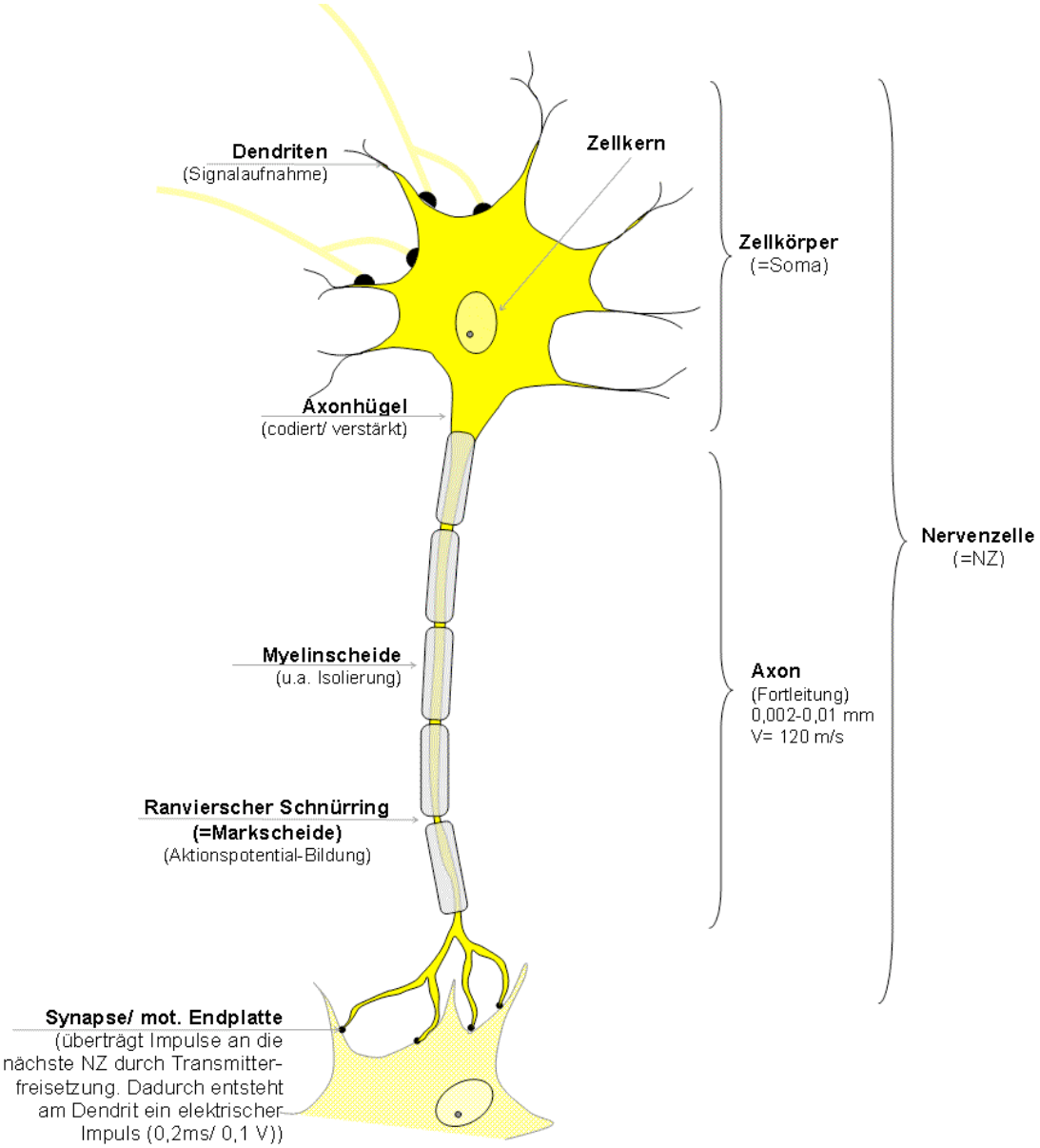

Bild 1: Schematische Darstellung einer Nervenzelle

Das Axon wiederum entspringt direkt aus dem Zytoplasma und ist ein kabelartiger Fortsatz, der sich an seinem Ende mannigfach verzweigt, um eine Vielzahl von Zielzellen zu erreichen. Das Axon übermittelt Information vom Zellkörper. Das Axon hat einen mikroskopischen Querschnitt von etwa 1 μm, verglichen dazu jedoch eine makroskopische Länge (<1 mm). Das längste Axon, das in einem menschlichen Körper vorkommt, kann bis zu einem Meter lang oder noch länger werden. Es sind dies die Axone des Ischiasnervs, welche vom Rückenmark bis zu den Zehen verlaufen. Das Axon ist umgeben von einer lipidreichen Biomembran, dem so genannten Myelin, welches das Axon isoliert. Es wird von den Schwannschen Zellen produziert.

Eine Schwannsche Zelle umhüllt in etwa 1 mm des Axons. Das Myelin wird durch die so genannten Ranvierschen Schnürringe unterbrochen. Somit ist das auch der Bereich zwischen zwei Schwannschen Zellen, die sich unter der Myelinschicht befinden. Ein Schnürring hat etwa eine Länge von einem µm. Diese Lücken sind der extrazellulären Flüssigkeit ausgesetzt. Diese Schnürringe spielen nun bei der Ausbreitung des Aktionspotentials eine wichtige Rolle. Die Impulse springen direkt von Schnürring zu Schnürring, anstatt sich über die gesamte Axonslänge depolarisieren zu müssen. Myelinisierte Axone übertragen Impulse etwa um das 10fache schneller als unyelinisierte Axone.

Am Ende des Axons sitzen nun die Synapsen [11], die, wie bereits erwähnt, für die Übertragung der Aktionspotentiale von einem Neuron zum anderen bzw. vom Neuron zu anderen Zellen (z. B. Muskeln) verantwortlich sind. Ein Neuron besitzt in etwa 10000 Synapsen. Ein Großteil von chemischen Synapsen befindet sich im Gehirn. Das Gehirn eines Kleinkindes beherbergt etwa 10^{16} Synapsen, während ein Erwachsenengehirn zwischen 10^{15} und $5 * 10^{15}$ Synapsen enthält. Der Vollständigkeit halber sei noch erwähnt, dass eine Nervenzelle üblicherweise viele Dendriten besitzt, jedoch nur ein Axon aufweist.

Wie auch alle anderen Zellen besitzt die Nervenzelle eine Zellmembran [19], deren Haupteigenschaft der Schutz der Zelle ist. Neuronen sind in einer Zellflüssigkeit eingelegt, und die Zellmembran verhindert die Durchmischung der inneren mit der äußeren Zellflüssigkeit. Entlang der Zellmembran findet man auch die eingebetteten Ionenkanäle. Sie kontrollieren und regeln den elektrochemischen Gradienten entlang der Zellmembran und erlauben den Ionentransport entlang derer. Üblicherweise sind die Ionenkanäle sehr spezifisch, d. h. sie sind nur durchlässig für eine bestimmte Ionenart, wie z. B. Kaliumionen oder Natriumionen. Es gibt aber auch unspezifische Ionenkanäle, die mehrere Arten von Ionen leiten, wie z. B. Kalium-, Kalzium- und Natriumionen.

2.3. Das Aktionspotential

Das Aktionspotential ist eine kurzzeitige Abweichung des Membranpotentials [19] einer Zelle von ihrem Ruhemembranpotential. Damit so ein Impuls also zustande kommen kann, ist im Neuron eine Spannungsänderung vonnöten. Aus der Konzentration der äußeren und inneren Zellflüssigkeit entsteht die Grundspannung einer Nervenzelle. Diese beträgt in etwa $-70\,mV$. Damit ein Aktionspotential zu Stande kommt, braucht die Zelle eine Spannung von etwa $20\,mV$. Dies ist der so genannte Threshold oder Schwellwert. Die Synapsen an den Dendriten sowie der Zellkörper sind für den Spannungsinput verantwortlich. Wichtig zu erwähnen ist, dass das Aktionspotential nach dem „Alles-oder-Nichts"-Prinzip entsteht. Hierbei werden die Inputs an allen Dendriten aufsummiert, und bei der Erreichung der $20\,mV$ Anhebung wird ein Impuls gefeuert. Liegt der Gesamtinput knapp unter dieser Schwellspannung, gibt es keinen Impuls. Damit sich dieses Aktionspotential überhaupt fortpflanzen kann, muss es zuerst entlang des Axons die Endsynapsen des Neurons erreichen. Erst dadurch ist die Weiterübertragung zu anderen Zellen möglich. Um eine derartige Fortpflanzung zu erreichen, ist eine Änderung des Spannungsverhältnisses der Zelle unumgänglich. Diese Änderung wird durch die bereits vorher erwähnten Ionenkanäle bewerkstelligt. Sie öffnen sich jedoch erst bei Erreichen der Schwellenspannung, also bei einer Anhebung um $20\,mV$. Das sukzessive Öffnen der Ionenkanäle bewirkt das Fortpflanzen des Aktionspotentials. Aufgrund dieser Vorgangsweise kann ein Impuls auf natürlichem Wege nicht verloren gehen. Unabhängig von der Stärke des auslösenden überschwelligen Reizes ist die Form des Aktionspotentials immer gleichförmig.

Im Folgenden wird nun anhand der Skizze der Ablauf eines Aktionspotentials beschrieben:

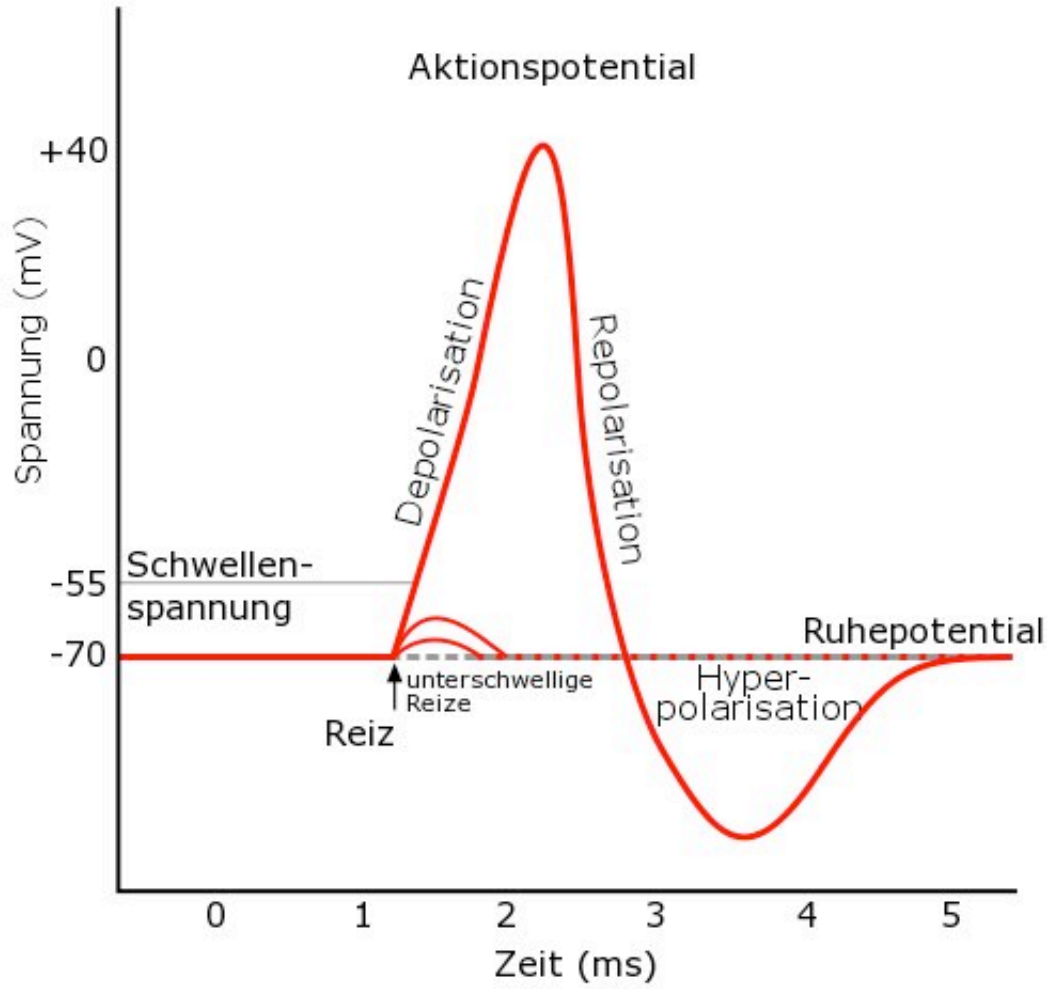

Bild 2: Schematische Darstellung: Ablauf eines Aktionspotentials

In der so genannten Ausgangslage herrscht das Ruhemembranpotential und die Zelle befindet sich in Ruhe. Für die jeweiligen Ionenbewegungen ist die Richtung und Stärke durch die elektrochemischen Triebkräfte bestimmt. Die Natriumkanäle sind in der Ausgangslage geschlossen, und nur bestimmte Kaliumkanäle sind geöffnet, weil die Kaliumionen das Ruhemembranpotential bestimmen.

In der Initiationsphase wird durch einen Reiz das Membranpotential bis zum Schwellenwert depolarisiert. Die Natriumkanäle beginnen sich zu öffnen. Diese bewirken einen raschen Transfer von Natriumionen in die Zelle. Weil diese positiv geladen sind, ändert sich der elektrochemische Gradient. Wenn das Membranpotential um 20 mV über das Ruhepotential angestiegen ist, kommt es zur raschen Depolarisation.

Die Zelle wird aufgrund der einströmenden Natriumionen immer weiter depolarisiert, wodurch sich immer mehr Kanäle öffnen und noch mehr Ionen einströmen können. Aufgrund des schnellen Anstiegs kommt es so zum Overshot bzw. zur Überschreitung während der Depolarisationsphase.

Noch bevor das Potentialmaximum erreicht ist, beginnen sich die Natriumionenkanäle bereits zu schließen. Auch die Kaliumionen kommen nun ins Spiel. Sie strömen jetzt aus der Zelle hinaus. Obwohl die Kaliumionenkanäle ihre Schwelle bei ähnlichen Werten haben, brauchen sie länger bis sie sich öffnen. Erst jetzt ist der Zeitpunkt hierfür gekommen. Während die Natriumionenkanäle bereits bis an ihr Maximum geöffnet sind, sind es die Kaliumionenkanäle erst bis zur Hälfte. Diese erreichen ihr Maximum erst, wenn die Natriumionenkanäle bereits fast zur Gänze wieder geschlossen sind. Daraus ergibt sich, dass sich das Natriummaximum kurz vor der Spannungsspitze befindet, das Kaliummaximum jedoch in der Phase der steilsten Repolarisation angesiedelt ist.

Während der Repolarisation gleicht sich das Potential nach und nach wieder dem Ruhepotential an. Die Kaliumkanäle schließen sich, was zu einer Stabilisierung des Ruhepotentials führt.

Vor allem bei Neuronen gibt es dann noch die Phase der Hyperpolarisation. Sie ergibt sich aus der noch erhöhten Kaliumleitfähigkeit, wodurch sich das Potential kurzzeitig noch näher beim Kaliumgleichgewichtspotential befindet. Das Membranpotential befindet sich also für eine kurze Zeit unter dem Ruhemembranpotential.

Nach einer Erregung gibt es eine kurze Zeitspanne, in der die Zelle nicht erregbar ist. Diese Zeit nennt man Refraktärzeit. Sie wird bestimmt durch die Dauer, welche die Ionenkanäle für die Wiederaktivierung benötigen. Während der Repolarisationsphase können die Natriumkanäle überhaupt nicht geöffnet werden. Dies ist dann die absolute Refraktärzeit, kurz nach dem Aktionspotential. Die ungefähre Zeitdauer beträgt in etwa $1 – 3\ ms$. Die Refraktärzeit ist unabkömmlich, da sie die Unidirektionalität, also die Weiterleitung des Aktionspotentials in nur eine Richtung, sicherstellt.

Ursprünglich vermutete man, dass die Synapsen an den Dendriten direkt anschließen, was jedoch nicht der Fall ist. Der Abstand zwischen den Synapsen und Dendriten ist etwa 20 *nm* breit und heißt synaptischer Spalt. In diesen werden die Neurotransmitter abgegeben. Aufgrund dieser verschwindend kleinen Größe ist eine Übertragung selbst durch hoch auflösende Mikroskope nicht möglich. Da bereits einige Male von Spannungsverhältnissen zwischen der intra- und extrazellulären Zellflüssigkeit gesprochen wurde, kann man ableiten, dass die Zellmembran die Eigenschaften eines Kondensators besitzt. Eine Zellmembran weist eine konstante Kapazität von 1 $\mu F/cm^2$ und einen nicht konstanten Widerstand von 1 – 50 $k\Omega/cm^2$ auf.

2.4. Die Ionenkanäle

Bis jetzt sprachen wir in den vorigen Abschnitten über die Fortpflanzung eines Aktionspotentiales entlang eines Axons. Weil ja die Fortpflanzung kein punktuell dauernder Vorgang ist, folgt daraus, dass die Spannungsverteilung einer Nervenzelle bei der Stimulation nicht überall die gleiche ist. Für diese Eigenschaft zeichnen sich die Ionenkanäle verantwortlich. Wie bereits erwähnt, unterscheidet man bei Neuronen zwei Sorten von Ionenkanälen: Natrium- und Kaliumkanäle. Außerhalb der Zellmembran ist die Konzentration von Natriumionen größer als die der Kaliumionen, wohingegen es sich im Zellinneren umgekehrt verhält. Bei einer Stimulation kommt es dann zu einer Konzentrationsänderung zwischen den beiden Ionentypen (Kalium- bzw. Natriumionen). Wie bereits berichtet, besitzen die Ionenkanäle ein unterschiedliches Öffnungs- bzw. Schließverhalten, welches von stochastischer Natur ist. Es kann also bei einem Aktionspotential weder a priori noch a posteriori gesagt werden, welche Ionenkanäle aktiviert werden. Unser Interesse gilt nun den Fragen, warum Ionenkanäle aufgehen, wie lange sie offen bleiben, wann sie wieder schließen. Diese Fragen kann man mit dem englischen Fachterminus „Gating“ umschreiben. Die Leitfähigkeit der Ionenkanäle hängt wesentlich von ihrem Milieu, ab in dem sie sich befinden. Überaus viele Ionenkanäle werden durch das Membranpotential gesteuert. So sind z. B. Natriumkanäle während des Ruhemembranpotentials nicht leitfähig. Es bedarf einer Depolarisation, um sie zu aktivieren. Manche Ionenkanäle können aber

auch durch physikalische Einflüsse wie Druck oder Vibrationen aktiviert werden. Unter allen Transportproteinen haben die Ionenkanäle die größte Durchflussrate.

2.5. Konzentrationsabhängige Gleichungen

2.5.1. Nernst Gleichung

Um die Zellmembranspannung ein wenig besser zu verstehen, gibt uns die Nernst Gleichung einen ersten Ansatz. Sie gibt die Spannung einer Zellmembran unter dem Einfluss einer einzigen Ionenart an. Die Idee von Nernst [20] war, dass man die elektrische Arbeit, die man benötigt, um n Mole von Ionen von der Konzentration c_1 in die Konzentration c_2 zu bringen, gleich ist mit der osmotischen Arbeit, die man benötigt, um diese Ionen vom Volumen V_1 in das Volumen V_2 zu komprimieren – unter Berücksichtigung der Gesetze der Gasdynamik.

Damit nun das Volumen dV komprimiert werden kann, benötigen wir die Arbeit $dW = p*dV$, wobei p der Druck ist.

Die Arbeit ist durch folgendes Integral gegeben:

$$\text{Gleichung 1: } W_{osmotisch} = -\int_{V1}^{V2} p\, dV$$

Nach einsetzen von $p.V = n.R.T$ in (1), wobei $R = 8.31111411\ J/(mol.K)$ (Gaskonstante) und T die absolute Temperatur ist, erhält man:

$$\text{Gleichung 2: } W_{osmotisch} = -\int_{V1}^{V2} \frac{n.R.T}{V} dV = -n.R.T.ln\frac{V_2}{V_1}$$

Die Konzentration ergibt sich aus der Division von Menge durch Volumen. Das heißt: $c_1 = n/V_1$ bzw. $c_2 = n/V_2$, daraus folgt:

$$\text{Gleichung 3: } W_{osmotisch} = n.R.t.ln\frac{c_2}{c_1}$$

Die benötigte elektrische Arbeit um die Ladung Q gegen die Spannung V zu bewegen, ergibt sich aus deren Produkt:

Gleichung 4: $W_{elektrisch} = Q.V$

Die Ladung pro Mol wird durch die Faraday-Konstante $F = 9.64845.10^4\ C/mol$ angegeben.
Mit der Formel

Gleichung 5: $Q = n.z.F$

wobei z die Valenz der verwendeten Ionengruppe darstellt (für Na^+-Ionen ist $z = 1$), erhält man:

Gleichung 6: $W_{elektrisch} = n.z.F.E$

Wenn man nun $W_{osmotisch}$ und $W_{elektrisch}$ gleichsetzt, erhält man die Nernst-Gleichung für die Spannung über eine Membran:

Gleichung 7: $$V = \frac{R.T}{z.F}\ ln\ \frac{c_2}{c_1}$$

Der Faktor $(R.T)/F$ beträgt bei Raumtemperatur ca. 25 mV.

2.5.2. Goldmann Gleichung

Neben den Natrium- und Kaliumionen existieren noch eine Reihe weiterer Ionengruppen, wie z. B. die Chlorionen. Diese Ionen beeinflussen die Spannung einer Membran nach verschiedenen Abhängigkeiten. Im Jahre 1943 postulierte Goldman [21] seine „Constant Field Theory“ , die auf folgenden Annahmen beruht:

1. Es bewegen sich Ionen in der Membran, unter dem Einfluss von elektrischen Feldern und Konzentrationsgradienten. Dies passiert auf demselben Prinzip, wie es auch in freien Lösungen der Fall wäre.
2. Die Ionenkonzentration am Rande der Membran ist proportional zu der in der angrenzenden wässrigen Lösung.
3. Der Gradient des elektrischen Potentials ist innerhalb der Membran konstant.

Aus diesen Annahmen erhält man die Goldmann-Gleichung, die es uns ermöglicht, die Spannung der Membran zu berechnen:

$$\text{Gleichung 8: } V = \frac{RT}{F} ln \frac{P_K[K]_o + P_{Na}[Na]_o + P_{Cl}[Cl]_i}{P_K[K]_i + P_{Na}[Na]_i + P_{Cl}[Cl]_o}$$

$[K]_i$, $[Na]_i$, $[Cl]_i$	Ionenkonzentration innerhalb der Membran
$[K]_o$, $[Na]_o$, $[Cl]_o$	Ionenkonzentration außerhalb der Membran
P_K, P_{Na}, P_{Cl}	Permeabilität (Durchlässigkeit) [*cm/sec*]

Zur Permeabilitätsberechnung kann man folgende Formel benutzen

$$\text{Gleichung 9: } P_i = \frac{u\beta RT}{\alpha F}, \quad i = K, Na, Cl$$

u	Mobilität des Ions innerhalb der Membran
β	Partitionskoeffizient zwischen der Membran und der wässrigen Lösung
α	Dicke der Membran

Die Ruhespannung der Membran errechnet man unter Zuhilfenahme der Goldmann-Gleichung nun folgendermaßen:

$$\text{Gleichung 10: } V = 58 \log \frac{10 + (0.03).460 + (0.1).40}{400 + (0.03).50 + (0.1).540} = -70\ mV$$

Für die Berechnung der Ruhespannung wurde das Tintenfischaxon des Hodgkin-Huxley-Experiments herangezogen. Hier gelten folgende Permeabilitäten:
$P_K : P_{Na} : P_{Cl} = 1 : 0.03 : 0.1$

2.5.3. Zellmembranen als elektrische Netzwerke

Um ein elektrisches Netzwerk für Zellmembranen aufzubauen, ist die sinnvolle Verknüpfung von Spannung, Stromstärke, Widerstand usw. notwendig.
Die Zellmembran weist eine konstante Kapazität auf, die aufgrund ihrer lipiden Doppelschicht mit $2\ \mu F/cm^2$ angenommen werden kann. Wegen der hohen Sensitivität des Öffnungsmechanismus der Ionenkanäle variiert der Widerstand jedoch sehr stark zwischen 1 bis $50\ k\Omega/cm^2$. Unser Ziel ist es nun, Voraussetzungen zu finden, um eine konstant gehaltene Membranspannung zu erzeugen und die Bedingungen zu erforschen, um einen unwillkürlich andauernden Stromimpuls zu generieren. Die Membranspannung V definiert sich wie folgt als Differenz des inneren vom äußeren elektrischen Potentials.

Zu Beginn lassen wir einen $1\ ms$ langen Stromimpuls mit einer Stärke von $1\ \mu A$ durch eine Teilfläche von $1\ cm^2$ durchlaufen. In diesem ersten Modell wird die Membran als ohmscher Widerstand modelliert. Wir wenden das Ohmsche Gesetz an $(V = R.I)$ und zeigen, dass eine Spannung von $2\ mV$ benötigt wird, um den gewünschten Stromimpuls zu erlangen.
Das zweite Modell bezieht sich nur auf die Kapazität der Zellmembran, die nun als Kondensator modelliert wird. Die Stromstärke gibt uns an, wie viele Ladungen pro Zeiteinheit durch die Zellmembran fließen.
Während des Impulses ist das Gefälle der Spannung konstant, woraus folgt: $1\ \mu A/\ 1\ \mu F = 1\ mV/ms$.

Um unser Modell nun zu vervollständigen, verbinden wir die beiden vorherigen Modelle miteinander. Die Teilfläche der Membran wird nun als Stromkreis mit einem Widerstand, Kondensator und einer kapazitiven Stromquelle angesehen. Daraus ergibt sich folgende Gleichung:

$$\text{Gleichung 11: } I = I_R + I_C = \frac{V}{R} + C.\ \frac{dV}{dt}$$

Nach Umformungen ergibt sich eine Differentialgleichung, mit der sich die Spannung der Zellmembran berechnen lässt. Wie man an der Formel erkennen kann, ist selbst bei sehr hoher Stromstärke das Zustandekommen eines Mindestimpuls von $20\,mV$, der ja für das Auslösen des Aktionspotentials erforderlich ist, quasi unmöglich. Von hoher Wichtigkeit ist auch die Erkenntnis, dass Spannung und Strom in einem nicht linearen Verhältnis stehen. Das heißt, dass für eine konstante Spannung ein nicht konstanter Strom benötigt wird und umgekehrt.

2.6. Das Hodgkin-Huxley Modell

Die Nobelpreisträger Sir Alan Lloyd Hodgkin und Sir Andrew Fielding Huxley gelten gemeinhin als Väter der Neurobiologie. Die ihrer Arbeit „A Quantitative Description of Membrane Current and its Application to Conduction and Excitation in Nerve“ beschreiben sie das Verhalten der Ionenkanäle bei der Übertragung von Aktionspotentialen, also den so genannten Gating-Mechanismus – es ist dies die meistpublizierte Arbeit der Welt [22-26]. Die mathematische Beschreibung – also die Modellierung – des Gating-Mechanismus gelang ihnen beim so genannten „Space Clamp Experiment“.

Das Ziel der Experimente von Hodgkin und Huxley (HH) war also herauszufinden, nach welchen Prinzipien oder Regeln der Ionenfluss in und aus dem Neuron während eines Aktionspotentials stattfindet. Am Beginn ihrer Experimente wussten sie, dass das Aktionspotential mit dem Eintritt von Natrium- und Kaliumionen zusammenhängt. Weiters wussten sie, dass die Rate und Amplitude des Aktionspotentials von der Natriumkonzentration außerhalb des Neurons bestimmt werden. Die Hauptannahme ihres Experiments war dann, dass der Membranstrom in einen Kapazitätsstrom und einen Ionenstrom geteilt werden kann. Der Kapazitätsstrom hängt von der Ionendichte an der Innen- sowie Außenseite der Membran ab. Der Ionenstrom ist abhängig vom Fluss der Natrium- und Kaliumionen durch die Membran.

Bei diesem Experiment legten Hodgkin und Huxley eine Tintenfischfaser in eine Flüssigkeit aus einem Natrium, Kalium, Chlor Gemisch ein. Durch Simulationsver-

suche wollten sie das Prinzip des Nervenerregungsmechanismus erforschen. Diese eben beschriebene Flüssigkeit nennt man auch Ringersche Flüssigkeit. Sie ersetzt in diesem Falle die extrazelluläre Flüssigkeit. Nun führten sie zwei unisolierte Silberdrahtelektroden der Länge nach in das Tintenfischaxon ein, welches an beiden Enden abgebunden war. Die Elektroden hatten einen Durchmesser von etwa *20 µm* und wurden etwa *20 – 30 mm* in das Neuron eingeführt. Während die eine Elektrode der Stimulation diente, wurde die andere zur Messung verwendet. Um eine konstante Sollspannung zu gewährleisten, wurde ein Stromgenerator mit Hilfe der Messelektrode gesteuert. Der hier benötigte Stimulusstrom weicht in seiner Form von der konstanten Form, die eine „technische Membran“ mit konstantem Ohmschen Widerstand und konstanter Kapazität aufweist, ab. Der durch die Ionenkanäle fließende Strom wird in I_{Na} (Natriumstrom), I_K (Kaliumstrom) sowie I_L (Leckstrom) unterteilt. Der Leckstrom gibt hier eine Zusammenfassung aller anderen Ionentypen an. Das Hodgkin-Huxley-Modell beschränkt sich also in der Beschreibung auf die zwei wesentlichen Ionenströme, induziert durch die Na^+ und K^+-Ionen. Der bereits erwähnte Leckstromanteil ist hierbei nur sehr klein.

Gleichung 12: $I = I_{Na} + I_K + I_L$

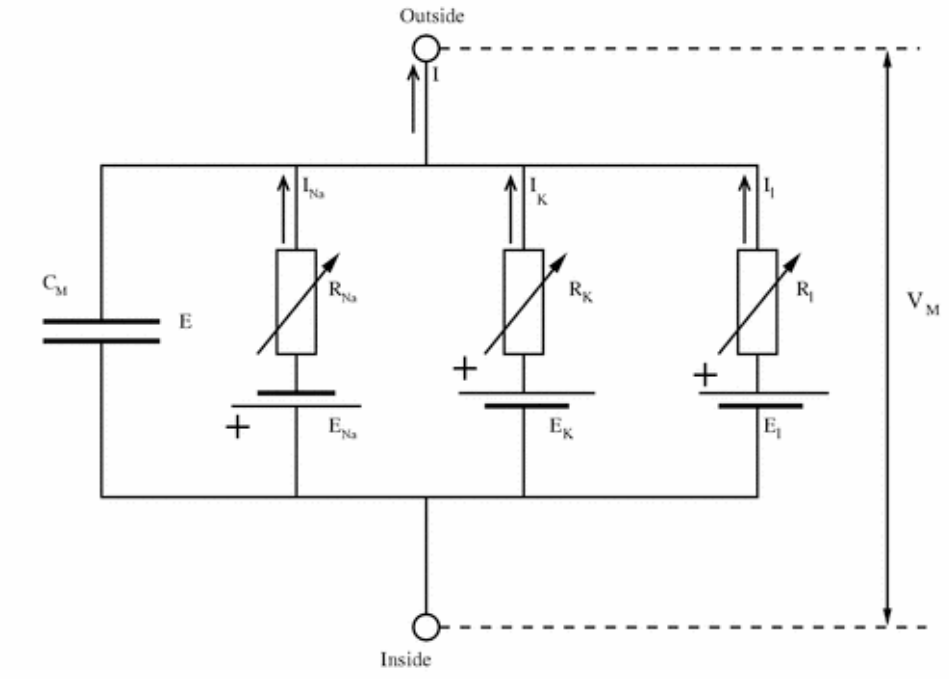

Bild 3: Schaltkreisdiagramm, das Stromflüsse in der Zellmembran eines HH-Modells beschreibt

Um die Ionenströme noch genauer zu bestimmen, führten Hodgkin und Huxley stochastische Größen *m*, *n* und *h* ein, um den Gating-Mechanismus der Ionenkanäle zu simulieren. Diese hängen von einer maximalen Leitfähigkeitskonstante ihres Ionentyps, der Spannung und einer stochastischen Größe ab. Im Folgenden die drei Gleichungen für die jeweiligen Ströme:

Gleichung 13: $I_{Na} = g_{Na}.m^3.h.(V- V_{Na})$

Gleichung 14: $I_K = g_K.n^4.(V-V_K)$

Gleichung 15: $I_L = g_L.(V-V_L)$

Für das Verständnis unserer Problemstellung ist die hier angeführte biologische Einführung mehr als ausreichend. Für eine detaillierte und über die hier angeführten Themen hinausgehende Auseinandersetzung mit der Materie kann auf die angeführte Literatur im Literaturverzeichnis verwiesen werden.

3. Neuron

3.1. Eine allgemeine Einführung

Das NEURON Simulation Environment [28-32] wurde entwickelt, um eine bequem zu nutzende und effiziente Umgebung zur Modellsimulation von biologischen aber auch künstlichen Neuronen zu schaffen. NEURON ermöglicht die Simulation von einzelnen aber auch von vernetzten Neuronen. Bei der Entwicklung wurde großer Wert darauf gelegt, in jeder Entwicklungsstufe die höchstmögliche algorithmische Effizienz und Robustheit zu gewährleisten. Gleichzeitig sollte dem Nutzer eine konzeptionelle Klarheit zur Verfügung gestellt werden. Der Benutzer sollte stets die Gewissheit haben, dass die Computersimulationen eine realitätsgetreue Implementierung und Berechnung seines konzeptionellen Modells darstellen. Das Einsatzgebiet von NEURON erstreckt sich weit über die Grenzen von stetigen Simulationen von Modellen einzelner Neuronen mit komplexen anatomischen und biophysischen Eigenschaften und beinhaltet auch die Simulation von unstetigen und hybriden Ereignissen die auch aus biologischen und künstlichen Neuronenmodelle kombiniert sein können.

NEURON ist besonders geeignet für Simulationen von experimentellen Daten, im Speziellen für jene Fälle, in denen Zellen mit komplexen anatomischen und biosphysikalischen Eigenschaften simuliert werden sollen.

3.2. Entstehungsgeschichte und Einsatzgebiete

Die Entwicklung von NEURON begann in den Laboratorien von John W. Moore an der Duke University, wo er gemeinsam mit Michael L. Hines eine Zusammenarbeit zur Entwicklung einer Simulationssoftware für Neurowissenschaften startete. Das Projekt profitierte von der nachhaltig überlegten Revision und den selektiven Erweiterungen, die sich aus den vielen Rückmeldungen der immer größer werdenden Nutzergruppe von Neurowissenschaftern ergaben. Die meisten dieser Wissenschafter verwenden Neuron als Hilfsmittel für die empirische Modellierung in ihren Forschungsstrategien.

Bis zum heutigen Tage ist NEURON schon in über 300 Forschungsartikeln als Tool zur Forschungssimulation genannt worden. Unter diesen Forschungsbereichen befinden sich Beschreibungen von Modellen, bestehend aus einzelnen Neuronen aber auch Modelle aus Netzwerken von Neuronen, mit den verschiedensten Eigenschaften:

- komplexe, verzweigte Topologien
- mehrere Ionenkanaltypen
- inhomogene Kanalverteilungen
- verschiedenste Ionenverteilungen
- sekundäre Botenstoffe und benutzerabhängige neuronale Plastizitäten
- etc.

3.2.1. Beispiele aus der Forschung

Auf der Zellebene wird NEURON unter anderem für die Erforschung folgender Themengebiete eingesetzt:

- Prä- und postsynaptische Mechanismen involviert in den synaptischen Übertragungen
- Funktionen der Dendritenarchitektur sowie der aktiven Membraneigenschaften
- Feuerung von Impulsen und deren Fortpflanzung in den Dendriten und Axonen
- die Auswirkungen der anatomischen und biophysikalischenVeränderungen
- funktionelle Genomik der Ionenkanäle
- extrazelluläre Stimulation

3.2.2. Einsatzgebiete in der Medizin

Die mit NEURON modellierten Neuronenmodelle wurden – um nur einige zu nennen - für folgende Einsatz- und Forschungsgebiete verwendet:

- Ursache der kortikalen und thalamischen Oszillationen
- Gap Junctions in den Neuronalen Oszillationen
- Informationsentschlüsselung in biologischen Netzwerken

- Empfindlichkeit der visuellen Orientierung
- Mechanismen der Epilepsie
- Wirkung krampflösender Medikamente

3.2.3. Bisheriger Einsatz zu Lehrzwecken an Universitäten

NEURON wird allmählich bedeutender im Unterricht der Neurowissenschaften. Die Simulationsumgebung wird heutzutage bereits – vor allem in den USA, aber auch weltweit – in den verschiedensten Studienrichtungen aller Jahrgänge eingesetzt. Viele dieser Kurse sind selbst gemacht, aber es gibt bereits Unterlagen, die im Druck erschienen sind (Moore and Stuart 2000), und die Autoren sind darauf bedacht, weitere Laborübungen bereitzustellen. NEURON ist deswegen sehr gut für den Bildungsbereich geeignet, weil spezielle Kenntnisse von numerischen Methoden oder ausgeprägte Programmierkenntnisse nicht vonnöten sind, um das Simulationstool produktiv einzusetzen. Überdies ist NEURON auf allen wichtigen Plattformen lauffähig, wie Windows, MacOS, UNIX/Linux. Auch auf haushaltsüblicher Hardware können in angemessener Zeit bereits hochqualitative Simulationen erstellt werden.

3.3. Die Funktionsweise von NEURON

Wie wir bereits wissen, läuft die Informationsverarbeitung im Gehirn als Interaktion von elektrischen und chemischen Signalen in und zwischen Neuronen ab. Daraus resultiert eine Vielzahl von räumlichen und zeitlichen Skalen, die sich aus den komplexen und non-linearen Mechanismen der Anatomie von Neuronen und deren Verbindungen ergeben. Folge dessen sind die Gleichungen, die Mechanismen im Gehirn beschreiben, üblicherweise nicht analytisch, und intuitive Ergebnisse zum Verständnis der Abläufe im Gehirn sind kaum möglich. Auch sind diese non-linearen und raumzeitlichen Eigenschaften sehr unähnlich im Vergleich zu sonstigen nichtbiologischen Systemen. Insofern ist eine sinnvolle Nutzung von vielen vorhandenen quantitativen oder qualitativen Modellierungstools, die diese komplexen Eigenschaften nicht berücksichtigen, nicht wirklich möglich.

NEURON wurde entwickelt, um eben diese beschriebenen Probleme lösen zu können. Erstens ermöglicht es die bequeme Erstellung von biologisch realistischen quantitativen Modellen von Gehirnmechanismen, und zweitens eine effiziente Simulation der Abläufe dieser Mechanismen. Das heißt jetzt nicht automatisch, dass der „Biologische Realismus“ einem unendlichen Detaillierungsgrad gleicht, sondern dass es dem Benutzer obliegt, die Details des Modells und auch der Simulation zu bestimmen – bis zum jeweiligen Grad der Notwendigkeit und nicht determiniert durch das Simulationsprogramm.

3.3.1. Zielgruppen

Dem Forscher bietet NEURON ein Hilfsmittel zur Vergleichsprüfung seiner Daten, zur Abschätzung von experimentell nicht zugänglichen Parametern und zur Entscheidungsfindung bei der Validierung von experimentellen Beobachtungen. Dem Theoretiker bietet das Tool eine Möglichkeit für den Test von Hypothesen und zur Bestimmung der kleinsten Unterschiede der anatomischen und biophysikalischen Eigenschaften die notwendig und hinreichend sind für die Auslösung von bestimmten Phänomenen. Dem Studenten bietet das Tool im Laborunterricht eine anschauliche Darstellung verschiedenster Funktionen in einer vereinfachten Form.

Die experimentellen Fortschritte der vergangenen 20 – 30 Jahre beeinflussten und förderten die quantitative Modellierung maßgeblich. Das Gebiet der Neurowissenschaften erfuhr einige bahnbrechende Entwicklungen in ihren experimentellen Verfahren. Hier seien nur einige genannt: hochqualitative elektrische Aufnahmen von Neuronen sowohl in vivo als auch in vitro durch Verwendung von Klemmen; gleichzeitige intrazelluläre Aufnahme von verbundenen pre- und postsynaptischen Neuronen; zeitgleiche Messung von elektrischen und chemischen Signalen; elektrische und optische Aufnahmen an mehreren Orten; quantitative Analyse der anatomischen und biophysikalischen Eigenschaften des gleichen Neurons; neue Medikamente wie Beta-Blocker; genetische Entwicklung von Ionenkanälen und Rezeptoren; Analyse der mRNA und der biophysikalischen Eigenschaften desselben Neurons; Knock-Out-Mutationen; uvam.

Diese und viele anderen Fortschritte zeichnen sich verantwortlich für die erstaunliche Entwicklung und die riesige Datenflut über Formulierungen von neuen Hypothesen von Gehirnfunktionen bei gleichzeitiger Sicherstellung der empirischen Basis für die biologisch realistischen quantitativen Modelle, die verwendet werden, um diese Hypothesen zu testen.

3.3.2. Vorteile von NEURON

NEURON bietet drei wesentliche Vorteile im Vergleich zu allgemeinen Simulationsprogrammen. Erstens ist es nicht nötig, das eigentliche Problem in eine andere Domäne zu konvertieren, sondern man hat die Möglichkeit, direkt mit den vertrauten Konzepten der neurowissenschaftlichen Anwendungen zu arbeiten. Zweitens stellt NEURON maßgeschneiderte Funktionen für die Simulation und graphische Darstellung von Lösungen von neurophysiologischen Fragestellungen dar. Drittens sind die internen Berechnungen derart optimiert worden, indem man spezielle Methoden und Tricks angewandt hat. Diese Methoden beziehen sich speziell auf die Struktur von Nervengleichungen.

3.4. Die Verwendung der NEURON Simulationsumgebung

Eine leistungsstarke und robuste Simulationsberechnung allein ist noch kein nützliches Simulationstool. Damit die Software wirklich hilfreich und nicht behindernd ist, ist es erforderlich, dass es eine adäquate und angepasste Bedienung zur Verfügung stellt. In diesem Kapitel wird die Bedienung von NEURON erörtert, sowie ein konkretes Beispiel eines einfachen Neurons dargestellt.

3.4.1. NEURONs Interpretersprache hoc

Der Kern von NEURON wird durch den so genannten `hoc`-Interpreter dargestellt. Hoc ist ein Gleitkommarechner mit einer C-ähnlichen Syntax, beschrieben durch Kernighan und Pike (1984). Der Interpreter wurde mit einer objekt-orientierten Syntax (außer Polymorphismus und Vererbung) ausgestattet, die es ermöglicht, abstrakte Datentypen zu verwenden. Dann gibt es noch zusätzliche Funktionen, die speziell die

Domäne der Neurowissenschaften bedienen, als auch Funktionen für die graphische Benutzeroberfläche. Mit Hilfe von hoc kann man relativ schnell und einfach kurze Programme für die meisten problemspezifischen Anforderungen programmieren. Der Interpreter selbst wird für die Ausführung der Simulation, die Anpassung der Benutzeroberfläche, die Anpassung von Parametern, zur Analyse von Daten aus Experimenten, zur Berechnung von neuen Variablen und für ähnliche weitere Einsatzgebiete verwendet.

Wie schon beschrieben haben Interpretersprachen oftmals einen erheblichen Performancenachteil gegenüber kompilierten Sprachen. NEURON umgeht dieses Problem, indem es rechenintensive Aufgaben von einem sehr effizienten vorkompilierten Code ausführen lässt. Zu diesen rechenintensiven Aufgaben gehört zum Beispiel die Integration der Kabelgleichung oder die Emulation der biologischen Mechanismen, die chemische und elektrische Signale hervorrufen.

Natürlich hat NEURON auch einen integrierten Texteditor. Man kann aber auch jeden beliebigen Text-/Programmiereditor verwenden, der ASCII-Dateien bereitstellen kann. Diese können von NEURON dann direkt ausgeführt werden.

3.5. Ein einführendes Beispiel

Im folgenden Beispiel will ich die konkrete Modellierung eines einfachen Nervenzellmodells (siehe **Bild 4**) zeigen.

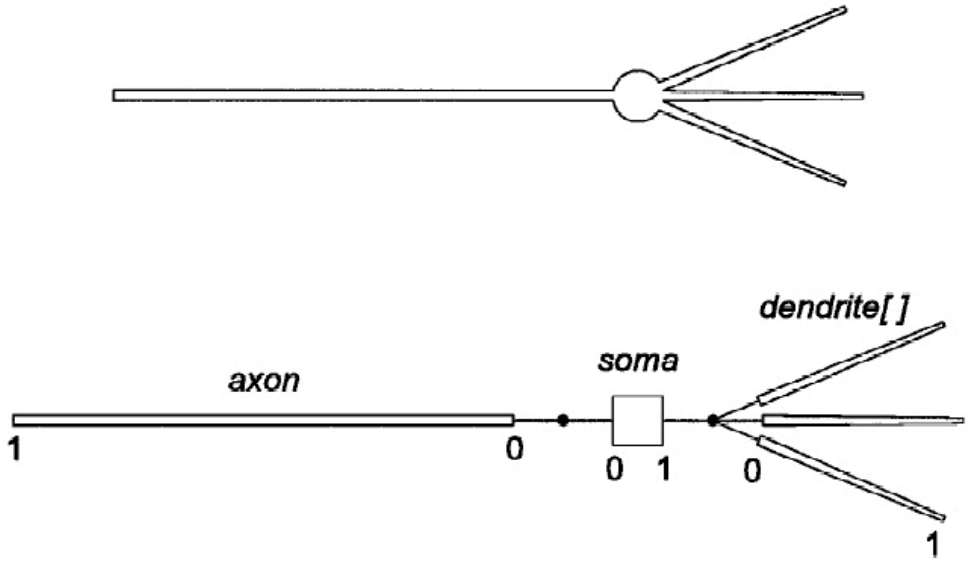

Bild 4: einfaches Nervenmodell: schematisch (oben); in NEURON (unten)

Die schematische Darstellung (oben) zeigt ein Neuron mit einem Zellkern (Soma), drei Dendriten auf der rechten Seite sowie einem unmyelinierten Axon auf der linken Seite. Das kugelförmige Soma hat einen Durchmesser von *50 µm*. Jedes Dendrit ist *200 µm* lang und hat am Ausgangspunkt (direkt am Soma) einen Durchmesser von *10 µm*. Am Endpunkt beträgt der Durchmesser nur noch *3 µm*. Das Axon ist *1000 µm* lang und weist einen Durchmesser von *1 µm* auf. Es wird eine Elektrode in das Soma eingeführt (in der Grafik nicht sichtbar), die der intrazellulären Stimulation mittels eines Stimulusstroms dient. Die untere Grafik zeigt die Darstellung desselben Modells, jedoch in Form einer Abbildung in NEURON. Die Erklärung hierzu findet man in der Beschreibung zur Modellierung.

Wie schon öfters erwähnt, ermöglicht NEURON dem Nutzer des Simulationstools sein Modell weiterhin aus neurophysiologischer Sicht zu sehen. Das heißt es gibt eine strikte Trennung der numerischen Angelegenheiten (wie etwa die Anzahl der räumlichen Segmente) und der Spezifikation der Morphologie und biophysikalischen Eigenschaften eines Modells. Diese wesentliche Eigenschaft von NEURON wird mit Hilfe von eindimensionalen Zylinderelementen als Grundbaustein für Zellmodel-

le erzielt. Diese Zylinderelemente können in jeder beliebigen Form miteinander verbunden und verzweigt werden. Sie müssen nur noch mit den spezifischen Eigenschaften versehen werden, die natürlich entlang des Verlaufes des Zylinders variieren können.

3.5.1. Erstellung des Grundgerüsts

Ein Neuron wie in **Bild 4** setzt sich aus verschiedenen anatomischen Gegebenheiten zusammen: In unserem Beispiel besteht es aus dem Zellkörper (Soma), drei Dendriten und einem unmyelinierten Axon. Der folgende Quellcode erstellt die Topologie dieses Modells in NEURON:

```
create soma, axon, dendrite[3]
connect axon(0), soma(0)
for i=0,2 { connect dendrite[i](0), soma(1) }
```

Das Programm beginnt mit der Erstellung (`create`) der jeweiligen Einheiten (Segmente), die mit den wirklichen anatomischen Gegebenheiten einer Zelle übereinstimmen. Mit Hilfe der `connect`-Funktion werden die Einheiten miteinander verbunden. Jedes Segment besitzt einen Positionsparameter, der sich von *0* bis *1* erstreckt. Dieser wird in runden Klammern nach dem Segmentnamen angeführt. Weil das Axon und die Dendriten jeweils von der anderen Seite des Zellkörpers entspringen (wie in **Bild 4** ersichtlich), wurden sie auch hier jeweils an gegenüberliegenden Seiten angehängt. Man kann prinzipiell an jeder beliebigen Stelle weitere Untersegmente anhängen, jedoch ist dies eher unüblich für andere Stellen als *0* oder *1* (Ausnahmefälle sind zum Beispiel Ausläufer an den Dendriten). Die `for`-Schleife ist nur eine abgekürzte Schreibweise für das Verbinden der Dendriten *1* bis *3*.

3.5.2. Hinzufügen der Eigenschaften

Im nächsten Schritt geht es darum, den einzelnen Segmenten ihre spezifischen anatomischen und biophysikalischen Eigenschaften zuzuweisen. Jedes Segment hat seine eigene Länge, Compartments, seinen Durchmesser, etc. Es gibt in NEURON verschiedene Schreibweisen um ein spezielles Segment zu initialisieren. Hier ist die bequemste und meist gebräuchliche Schreibweise verwendet worden:

```
// Anatomische und biophysikalische Eigenschaften zuweisen

soma {
      nseg = 1    // Anzahl der Compartments
      L = 50      // [μm] Länge
      diam = 50   // [μm] Durchmesser
      insert hh   // Standard-Hodgkin-Huxley Ströme hinzufügen
      gnabar hh = 0.5*0.120 // max. HH Natrium-Leitwert
}

axon {
      nseg = 20
      L = 1000
      diam = 1
      insert hh
}

for i=0,2 dendrite[i] {
      nseg = 5
      L = 200
      diam(0:1) = 10:3  // Durchmesser der Dendriten wird kleiner
      insert pas        // passiver Strom
      e_pas = -65       // [mv] Ruhepotential für pass. Strom
      g_pas = 0.001     // [siemens/cm2] Leitwert für pass. Strom
}
```

Mit Hilfe des `nseg` Parameters kann ein Segment in räumlicher Ebene in mehrere so genannte Compartments aufgeteilt werden. In unserem Beispiel bleibt der Zellkern ein einziges Compartment (`nseg = 1`), während das Axon in *20* Compartments aufgeteilt wird und die Dendriten jeweils in fünf.

Das Axon in unserem Beispiel wird durch einen Zylinder dargestellt, der einen gleich bleibenden Durchmesser entlang der gesamten Länge aufweist. Das Soma, das eigentlich eine Kugel ist, wird ebenfalls durch einen Zylinder dargestellt. Dieser hat dieselbe Oberfläche wie die zugehörige Kugel. Somit ist die Approximation an einen Zylinder keine nennenswerte Fehlerquelle. Wenn man chemische Signale (etwa intrazelluläre Ionenkonzentrationen), adäquat modellieren wollte, müsste man neben der Oberfläche auch das Volumen dementsprechend approximieren. Die Dendriten werden mit steigender Entfernung vom Soma dünner. Sie sind auch zu lang, um durch ein einziges Compartment abgebildet zu werden, vorallem, weil sie ja keinen konstanten Durchmesser aufweisen. Mit Hilfe des Befehls `diam(0:1) = 10:3` und den zugehörigen Bereichsvariablen kann der Durchmesser stetig kleiner gemacht werden.

Unser Modell beinhaltet im Soma und im Axon jeweils Hodgkin–Huxley–Kanäle für den Natrium-, Kalium- sowie für den Leckstrom. Die Dendriten weisen konstante, lineare Ionenströme auf. Mit Hilfe der `insert`-Funktion werden die diesbezüglichen Mechanismen den jeweiligen Segmenten zugewiesen. NEURON bietet beide beschriebene Ionenkanäle bereits standardmäßig an. Man kann dann die einzelnen Spezifikationen für die Membranmechanismen separat verändern. Dies haben wir auch in unserem Beispiel getan mit dem Natriumkanal im Soma (`gnabar_hh`), mit den Ionenströmen sowie mit dem Ruhepotential der Dendriten (`g_pas` und `e_pas`). Details zu diesen Membranmechanismen werden später noch genauer beschrieben.

3.5.3. Platzieren der Elektrode

Jetzt ist es an der Zeit, um eine Elektrode im Soma zu platzieren um einen Stimulusstrom zu ermöglichen.

```
objref stim
soma stim = new IClamp(0.5) // paltziert in der Mitte des Zellkör-
pers
stim.del = 1              // [ms] Verzögerung
stim.dur = 0.1            // [ms] Stimulationsdauer
stim.amp = 60             // [nA] Amplitude
```

Mit Hilfe dieser Befehlszeilen wurde nun eine Stromelektrode in der Mitte des Somas platziert (Position *0.5*). Die Stimulation beginnt nach einer Verzögerung von *1 ms*, dauert *0.1 ms* lang und hat eine Amplitude von *60 nA*. Diese Stimulationselektrode ist ein so genannter Punktprozess, welcher in einem späteren Kapitel nochmals ausführlicher erklärt wird. Auch gibt es verschiedene Arten von vorgefertigten Elektroden, die ebenfalls später beschrieben werden.

3.5.4. Setzen der Simulationsparameter

Jetzt ist das Modell ausreichend beschrieben, und die wichtigsten Parameter wurden spezifiziert. Es fehlen nur noch die Simulationsparameter, welche den zeitlichen Ablauf der Simulation beschreiben, sowie der Quellcode, der die Simulation schließlich ausführt. Dies passiert üblicherweise in zwei Schritten. Zuerst wird das Modell initialisiert, das heißt die Membranpotentiale sowie die eingefügten Mechanismen werden aktiviert (Ionenkanalzustände, Ionenkonzentrationen, extrazelluläre Potentia-

le, ...). Im zweiten Schritt wird die eingebaute Integrationsfunktion `fadvance()` wiederholt aufgerufen. Diese speichert, zeichnet oder berechnet Funktionen der gewünschten Ausgabewerte für jeden Schritt. Das heißt auch, dass eine Integrationsschrittweite angegeben werden muss.

```
dt = 0.05          // [ms] Zeitintervall der Integration
tstop = 5          // [ms] Endzeitpunkt der Simulation
finitialize(-65)   // [mV] Initialisierung Membranpotential,
                   // Zustandsvariablen und Zeit

proc integrate() {
            print t, soma.v(0.5) // gibt Startzeit und Membranpoten
                                    tial aus

      while (t < tstop) {
            fadvance()  // Integration

            // Funktionen zum Speichern oder Plotten hier einfügen

            print t, soma.v(0.5) // aktuelle Zeit und Membranpot.

            // Hier könnten Befehle zur Modellveränderung eingefügt
            // werden.
      }
}
```

Die Variablen `dt` und `tstop` sind globale Variablen, die für die Integrationsschrittweite sowie den Endzeitpunkt der Simulation vorgesehen sind. Mittels der Funktion `finitialize()` wird die Zeitvariable `t` auf `0` gesetzt sowie das Membranpotential des Modells auf `v = -65 mV` initialisiert. Weiters werden mit dem Funktionsaufruf auch die HH-Kanäle und deren Zustandsvariablen `m`, `n` und `h` in den Ruhezustand bei `v = -65 mV` initialisiert. Natürlich kann die Initialisierung auch von Hand vorgenommen werden, wenn die Standardinitialisierung `durch finitialize()` nicht ausreicht.

Die `while()`-Schleife ruft die Funktion `fadvance()` so lange auf, bis `t >= tstop` wird. Bei jedem Aufruf werden die Modellgleichungen über das Intervall dt integriert, sowie `t` um `dt` erhöht. Auch werden nach jedem Aufruf von `fadvance()` die aktuellen Werte der Zeit sowie des Membranpotentials in der Mitte des Somas ausgegeben.

Wenn man oben geschriebene Zeilen nun durch den NEURON Interpreter laufen lässt, wird das Modell erstellt und zwar initialisiert, aber noch nicht ausgeführt. Die `integrate()`-Funktion ist ja nur einmal definiert worden (mit Hilfe des `proc` Schlüsselwortes). Wenn man die Funktion `integrate()` dann in der NEURON-Kommandointerpreterzeile ausführt, läuft die Simulation für *5 ms* mit *5 µs* Schrittweite.

3.5.5. Segmentvariablen

Es gibt drei Variablen in NEURON, die sich auf Segmente beziehen. Dies sind die Länge `L`, der Compartmentparamter `nseg` sowie der zytoplasmatische Widerstand `Ra` *(Ωcm)*. Die Länge und der Widerstand beeinflussen die Struktur der Differentialgleichungen, die das Modell beschreiben, überhaupt nicht. Den Wert für `Ra` hat man in unserem Beispiel nicht extra bestimmt, weil man üblicherweise annimmt, dass das Zytoplasma innerhalb einer Zelle gleichförmig ist. Der Standardwert von `Ra` ist 35.4 *Ωcm*. Wie auch `nseg` und `L` kann er in der gleichen Art und Weise verändert werden (z. B. 200 *Ωcm* bei Neuronen von Säugetieren).

Über den Parameter `nseg` kann der Benutzer die Anzahl der Compartments anpassen, ohne andere anatomische oder biophysikalische Eigenschaften verändern zu müssen. Jedoch muss man darauf achten, falls die Parameter innerhalb eines Segments variieren, dass die Veränderung der Compartments Auswirkungen haben kann. Diese Problematik wird später noch weiter erklärt.

3.5.6. Bereichsvariablen

Viele zellulare Eigenschaften sind Funktionen des Positionsparameters `x`. In unserem Beispiel ist das der Durchmesser der Dendriten, der sich von *10 µm* auf *3 µm* linear verkleinert. Zur Unterstützung dieser Funktionen stellt NEURON eine spezielle Methode zur Verfügung, die so genannten Bereichsvariablen. Weitere typische Einsatzgebiete von Bereichsvariablen wären das Membranpotential `v` und ionische Strömungsleitwerte, wie etwa der maximale Hodgkin-Huxley Leitwert `gnabar_hh` (*siemens/cm*^{2}). Bereichsvariablen ermöglichen es, die Beschreibung der Komponenten von der Segmentierung zu trennen. Die Zuweisung der Bereichsvariablen kann auf

zwei Arten erfolgen. Die einfache und gebräuchlichere Variante ist die Zuweisung einer Konstante `etwa soma.diam = 50`. Das bedeutet, dass das Soma über die gesamte Länge einen konstanten Durchmesser hat.

Wenn sich nun eine Einheit über den Verlauf seiner Länge verändert, dann sieht die Zuweisung folgendermaßen aus: `bereichsvar(von : bis) = a : b`. Die Ausdrücke `a` und `b` entsprechen jeweils den Werten an den Positionen von und bis. Der Bereich dazwischen wird über die Segmente linear interpoliert. Die Positionsangaben von und bis sind folgender Auflage unterworfen `0 <=` von <= bis <= 1. Das heißt, dass von und bis sich in einem Bereich von `0` und eins befinden müssen. Mit Hilfe der linearen Interpolierung werden die Werte in den Segmentmitten berechnet.

In unserem Beispiel werden die Dendriten wie folgt beschrieben: `diam(0:1) = 10:3` mit `nseg = 5`. Daraus ergeben sich fünf Segmente mit ihren Mittelpunkten bei `x = 0.1, 0.3, 0.5, 0.7` und `0.9`. Die Durchmesser an besagten Stellen betragen dann `9.3, 7.9, 6.5, 5.1` und `3.7`.

Um nun den Wert an einer bestimmten Position zu erhalten kann man mit `bereichsvar(x)` mit `0 <= x <= 1` auf die Variable zugreifen. Zu beachten ist aber, dass hier nicht der interpolierte Wert für die Position x retourniert wird, sondern der Wert in der Mitte jenes Segmentes zu dem `x` gehört. Bei `diam(0.25)` würde etwa `7.9` retourniert werden, da sich `x = 0.25` bereits im zweiten Segment befindet und dessen Wert bekanntlich `7.9` ist. Wenn man bei der Abfrage die runden Klammern weglässt, entspricht das dem Wert `x = 0.5`, also der Mitte der gesamten Einheit.

Im Bereich der Bereichsvariablen gibt es noch eine Spezialform der `for`-Schleife: `for (var) stmt`. In diesem Fall besteht `var` aus genau jenen Werten, die der Mitte der Segmente einer Einheit entsprechen inkl. 0 und 1 (Beginn und Ende einer Einheit). Der folgende `hoc`-Code würde das jeweilige Membranpotential an der zugehörigen physikalischen Position ausgeben:

```
dendrite[0] for (x) print x*L, v(x)
```

In NEURON würde nun an den Positionen
`x = { 0, 0.1, 0.3, 0.5, 0.7, 0.9 } * L` (in μm) das zugehörige Membranpotential ausgegeben werden.

3.5.7. Geometrieeinstellungen

In NEURON gibt es zwei Möglichkeiten, die Geometrie einer Einheit zu beschreiben. Unser Beispiel nutzte die deskriptive Methode, bei welcher man einfach Werte für die Länge und den Durchmesser zugewiesen hat. Dies ist auch die gebräuchlichste Art der Eingabe, wenn die dreidimensionale Form und Ausprägung irrelevant ist.

Wenn man aber ein Modell, basierend auf anatomischen Daten, nachbilden will und die dreidimensionale Darstellung eine Rolle spielt, ist es angebracht, die 3D-Eingabemethode zu verwenden. Hierbei erstellt man eine Liste von (`x, y, z, durchmesser`)-Koordinaten die aus zumindest zwei Punkten bestehen muss. Die Länge und der Durchmesser werden automatisch aus der Liste berechnet und es ist naheliegend, die Punkte in der Liste der Reihe nach zu ordnen. Erwähnenswert ist noch, dass die Anzahl der Punkte zur Beschreibung der Form keine Auswirkung auf die Segmente `nseg` hat und auch die Simulationsgeschwindigkeit nicht beeinflusst.

3.5.8. Biophysikalische Mechanismen

Mittels des `insert`-Befehls kann man einer Einheit biophysikalische Mechanismen hinzufügen. Diese sind dann für die elektrischen und chemischen Signale verantwortlich. Entlang der Zellmembran finden sich mehrere solcher elektrischen und chemischen Signale. Zu diesen Mechanismen zählen unter anderem die Ionenkanäle, wie zum Beispiel die Hodgkin-Huxley-Kanäle. Man beschreibt sie mit Strom pro Fläche und Leitwert pro Fläche. Diese Beschreibungsform ist jedoch nicht auf alle Signalquellen applizierbar. Synapsen und Elektroden beschreibt man nämlich am besten über den absoluten Strom in *Nanoampere* und den absoluten Leitwert in *Microsiemens*. Bei diesen Mechanismen spricht man dann von den sogenannten Punktprozessen.

In unserem Beispiel haben wir in der Mitte des Somas eine Stromklemme eingefügt. Dafür haben wir zuerst mit Hilfe des Befehls `objref stim` ein Objekt deklariert. Im nächsten Schritt haben wir dann eine Stromklemme - also eine neue Instanz eines `IClamp`-Objekts - zugewiesen. Der mitgegebene Wert in Klammern (hier 0.5) entspricht der Position. Vor dem `stim`-Objekt muss die Einheit angegeben werden, der die Klemme zugewiesen werden soll: in unserem Fall „`soma`“. Wenn ein Punktprozess nicht mehr referenziert ist, d. h., wenn er keiner Einheit mehr zugeordnet ist, wird er automatisch von seiner aktuellen Position entfernt und gelöscht. In unserem Beispiel würde ein erneutes Aufrufen von `objref stim` genügen, um die Stromquelle vom Soma zu entfernen. Weiters ist es auch möglich, die Stromklemme mit ihren gesamten Einstellungen auf eine andere Position zu setzen. Der Befehl `dendrite[1] stim.loc(0.5)` würde in unserem Beispiel eine Repositionierung der Stromklemme auf die Mitte der zweiten (mittleren) Dendrite bewirken. Alle Einstellungen, wie etwa Stimulationsdauer und –stärke, würden erhalten bleiben.

In NEURON ist es möglich, viele dieser Stimulationsmechanismen und Punktprozesse gleichzeitig einzusetzen. Jede Einheit kann eine Vielzahl dieser Mechanismen enthalten. Der wesentliche Unterschied zwischen den Stimulationsmechanismen (wie etwa den HH-Kanälen oder anderen Ionenkanälen) zu den Punktprozessen ist, dass man mehrere gleichartige Punktprozesse pro Position vergeben kann. Das heißt, es ist ohne weiteres möglich, zwei (oder mehr) identische Stromklemmen an der exakt gleichen Position anzubringen, während man einen HH-Kanal-Mechanismus nur einmal pro Einheit einfügen kann.

Wenn man diese Simulationsmechanismen oder Punktprozesse anpassen oder verändern will, stellt NEURON eine eigene Sprache dafür zur Verfügung. Diese Sprache heißt NMODL, und sie ermöglicht dem Benutzer die Erstellung von Gleichungen für Ionenkanalprozesse o. ä. Der zugehörige NMODL-Übersetzer erstellt daraus ein lauffähiges C-Programm, das kompiliert wird und anschließend unter NEURON zur Verfügung steht. Weitere Informationen zu NMODL finden sie auf den Webseiten von NEURON [3], [4] oder in [34].

3.6. NEURONs graphische Oberfläche

NEURON bietet nicht nur die Steuerung und Eingabe der Simulationsumgebung über eine textbasierte Kommandozeile sondern stellt auch eine umfangreiche graphische Oberfläche zur Verfügung. Mit Hilfe dieser graphischen Oberfläche kann man ohne auch nur eine Zeile Code schreiben zu müssen, ziemlich einfach verschiedene Menüs generieren und anzeigen lassen, Parametereditoren verwenden (zur Anzeige und Veränderung von Daten), Graphen der verschiedensten Einstellungen, Parameter und Statusvariablen darstellen lassen und auch das modellierte Neuron graphisch darstellen lassen. Mit Hilfe der anatomischen Ansicht kann man so genannte Space-Plots erstellen, welche die Mechanismen und Punktprozesse anzeigen und auch zeigen, wo diese platziert sind.

Diese graphische Darstellung bringt große Vorteile mit sich, weil man sich das modellierte Neuron ansehen kann und man somit eine gute Kontrolle hat, ob das, was man vorher eingegeben hat, auch tatsächlich dem Neuron entspricht, das man sich vorgestellt hat. Auch bewahrt man mit Hilfe der graphischen Darstellung ständige Kontrolle über den Simulationsablauf, weil Veränderungen in den Berechnungen sofort auch graphisch angezeigt werden (im Vergleich zum mühsamen `print`-Befehl, der stets extra aufgerufen werden müsste).

Auf eine graphische Möglichkeit die Topologie eines Neurons zu erstellen wurde verzichtet. Sehr einfache Modelle lassen sich ohnedies ganz flott mit Hilfe des `hoc`-Codes beschreiben, sodass ein graphischer Editor unnötig wäre. Komplexere Modelle wiederum lassen sich nur schwer mittels eines graphischen Editors erstellen sondern werden meist durch komplexe Algorithmen beschrieben, welche dann mit Hilfe des `hoc`-Interpreters die Topologie des Modells automatisch generieren. Wenn man zum Beispiel sehr realistische biologische Modelle simulieren will, bestehen diese aus vielen hunderten oder sogar tausenden von Einheiten. Deren Eigenschaften und Dimensionen sowie Verbindungen untereinander sind meist in Tabellen organisiert. Mit Hilfe von `hoc`-Prozeduren kann man diese Tabellen einlesen und ohne weitere Benutzerinteraktionen Einheiten aus den Tabellendaten erstellen lassen (`create`) und miteinander verbinden (`connect`).

3.7. NEURONS objekt-orientierte Syntax

Für den Umgang mit größeren Netzwerken und gleichartigen Einheiten bietet der NEURON Interpreter einige syntaktische Konstrukte, die für die Vereinfachung und bessere Verwaltung gedacht sind. Für die Verwaltung von gleichartigen Einheiten, wie zum Beispiel eine Gruppe von Dendriten, bietet NEURON eine spezielle Datenklasse, genannt `SectionList`. Die Erstellung und Verwendung der `SectionList` fügt sich nahtlos in die anderen Befehle von NEURON ein. Nachdem die Liste erstellt wurde, kann sie aber viel effizienter wiederverwendet werden. Für jede Iteration, die auf alle gruppierten Elemente zutrifft, braucht man nur die `SectionList` zu durchlaufen und keine komplizierten Schleifen zu konstruieren, welche aus vielen einzelnen Befehlen bestehen und immer wieder dieselben Arbeitsschritte ausführen (Elemente auswählen, eigentlichen Befehl ausführen). Mit Hilfe des folgenden Beispieles sei das `SectionList`-Konstrukt veranschaulicht.

```
objref alldend
alldend = new SectionList()
forsec „dend“ alldend.append()
```

Der Befehl `forsec` iteriert über alle Elemente der Liste. Der `append`-Befehl fügt alle Einheiten zur Liste hinzu, die in der Zeichenkette „`dend`“ enthalten. Nun kann man das Objekt `alldend` mit allen seinen Einheiten verwenden. Der Befehl `forsec alldend print secname()` würde nun die Namen der Einheiten ausgeben. Für unser angeführtes Beispielprogramm würde die Ausgabe folgendermaßen lauten:

```
dendrite[0]
dendrite[1]
dendrite[2]
```

Natürlich hätte man in diesem einfachen Fall eine simple `for`-Schleife einsetzen können, jedoch wird augenscheinlich, dass bei der Verwaltung von mehreren Einheiten eine eindeutige Schreibersparnis und Erhöhung der Lesbarkeit zu erzielen ist.

Für die Erstellung von Netzwerken aus vielen einzelnen ähnlichen Neuronen gibt es ein syntaktisches Konstrukt, wie auch in den meisten Programmiersprachen: das Template. Im folgenden Beispiel erstellen wir unsere bereits verwendete Zelle bestehend aus Soma, drei Dendriten und einem Axon mit Hilfe eines Templates.

```
begintemplate Zelle1
      public soma, dendrite, axon
      create soma, dendrite[3], axon

      proc init() {
            for i=0,2 connect dendrite[i](0), soma(0)
            connect axon(0), soma(1)
            axon insert hh
      }
endtemplate Zelle1
```

Wenn man nun eine Instanz dieses Templates erstellt wird zuerst die `init`()-Prozedur aufgerufen, welche die Dendriten und das Axon automatisch mit dem Soma verbindet. Es wird auch ein Hodgkin-Huxley-Ionenkanalmechanismus ins Axon eingefügt. Natürlich kann man innerhalb der `init`()-Prozedur auch weitere Einstellungen und Spezifikationen tätigen. Die `init`()-Prozedur entspricht demnach einem Konstruktor in anderen objektorientierten Programmiersprachen. Wenn man auf Variablen von außerhalb zugreifen will, muss man sie im Bereich `public` auflisten. Nachdem die `init`()-Prozedur nicht im `public` Bereich steht, kann man sie auch nicht von außerhalb nochmals aufrufen. Eine Reinitialisierung durch den Nutzer ist somit nicht möglich. Die öffentlich zugänglichen Variablen (`soma`, `axon`, `dendrite`) referenziert man wie auch bei anderen objektorientierten Programmiersprachen mit einem Punkt (z. B. `NeueZelle.axon`).

Der größte Vorteil dieses Templates ist aber die Möglichkeit, viele gleichartige Zellen auf einmal zu erstellen und auch zu initialisieren. Folgendes Beispiel konstruiert ein Feld mit 5 x 5 Zellen:

```
objref Zellen[5][5]
for i=0,4 for j=0,4 Zellen[i][j]=new Zelle1()
```

Hiermit haben wir nun 25 gleichartige Neuronen des Typs Zelle1 erstellt, auf die man individuell über das Objekt `Zellen` zugreifen kann. In diesem Beispiel ist `Zellen[2][3].soma.v(0.5)` die Spannung in der Mitte des Somas der `Zelle[2][3]`. Natürlich kann man die Variablen nicht nur lesen sondern auch verändern.

An diesem Beispiel sieht man, dass Templates eine gute Möglichkeit bieten, große Zellenetzwerke zu erstellen. Es ist jedoch dem Benutzer überlassen, dass diese Templates in der Form miteinander verknüpft werden, dass eine logische Hierarchie entsteht.

3.8. Der interne Aufbau von NEURON

Die künstlichen Neuronen werden in NEURON im Vergleich zu deren biologischen Pendants relativ simple abgebildet. Daraus folgt, dass die Simulation dieser Neuronen mithilfe der diskreten Event-Methode bewerkstelligt werden kann. Diese ist viele hunderte Male schneller als vergleichbare numerische Integrationsmethoden. NEURON führt Berechnungen nur dann aus, wenn Events auftreten. Wenn ein Event auftritt, werden alle Statusvariablen neu berechnet und zwar aus den Statusvariablen und der Zeit t0 des vorherigen Events. Dies führt dazu, dass die kumulierte Rechenzeit proportional zur Anzahl der Events ist und unabhängig von der Anzahl an Zellen, Verknüpfungen oder der Zeit. Das bedeutet, dass die Berechnung von 50000 Impulsen in einer Stunde bei 100 Zellen genau so lange dauert wie, die Berechnung von 50000 Impulsen in einer Sekunde bei 10 Zellen.

NEURON stellt drei verschiedene Klassen von Modellen bereit. Die einfachste Feuerungsklasse ist IntFire1, ein einfacher Integrator der eingehende Events als gewichtete Deltafunktionen behandelt. Wenn so eine `IntFire1`-Zelle ein Eingangsevent mit der Gewichtung `w` erhält, erhöht sich das Membranpotential m sofort proportional zum Wert von `w` und vollführt anschließend einen Abfall Richtung 0 mit der Zeitkonstante `tm`.

Die Feuerungsklasse `IntFire2` funktioniert dem Verhalten von biologischen Neuronen entsprechend etwas ähnlicher. Hier wird zum Membranpotential m ein synaptischer Strom `i` integriert. Wenn eine `IntFire2`-Zelle ein Eingangsevent erhält, erhöht sich der synaptische Strom proportional zum synaptischen Gewicht. Anschließend fällt `i` in Richtung Ruhezustand `ib` mit seiner eigenen Zeitkonstante `ts` mit `ts > tm` ab. Ein einzelnes Eingangsevent führt zu einem sukzessiven Anstieg des Membranpotentials `m` mit einer verzögerten Erreichung des Höhepunktes (im Gegensatz zu `IntFire1` wo der Höhepunkt sprunghaft erreicht wird). Die Feuerungsrate entspricht `i/tm` wenn `i>1` und `ts>tm`.

`IntFire2`-Feuerungsklassen können eine große Bandbreite von Verbindungen zwischen Eingansevents und Feuerungsraten simulieren, jedoch produzieren alle Eingangsevents eine Rückmeldung mit derselben Kinetik, unabhängig davon, ob diese stimulierend oder hemmend wirken. In biologischen Neuronen sind die stimulierenden Reize jedoch meist schneller als die hemmenden. Dies führte zur Entwicklung der `IntFire4`-Feurungsklasse. Diese integriert zwei synaptische Stromkomponenten mit verschiedenen Dynamiken, je nachdem, ob die Eingangsevents stimulierend oder hemmend wirken. Stimulierende Eingangsevents werden sofort zu einem stimulierenden synaptischen Strom `e` addiert, analog zu `IntFire2`. Eine hemmender synaptischer Strom hingegen wird bei `IntFire4` durch ein Reaktionsschema beschrieben, das einem langsameren Anstieg des Membranpotentials und einem noch langsameren Abfall desselben entspricht.

Diese 3 Feuerungsklassen sind nicht die einzigen Arten von künstlichen Neuronen, die in NEURON simuliert werden können. Die einzige Vorraussetzung für die Simulation mit diskreten Events ist, dass alle Variablen eines Modellneurons von einem Initialzustand aus berechnet werden können. Wenn man also spezielle Anforderungen hat, kann man zusätzliche Neuronen-/Feuerungsklassen hinzufügen. Diese werden mit der NMODL Sprache beschrieben und in NEURON integriert.

3.8.1. Interne Darstellung und Funktionsweise eines Neuronennetzwerkes

In NEURON sind die so genannte `NetCon`-Klasse und das Event-Delivery-System für die Verwaltung der synaptischen Kommunikation zwischen den Modellneuronen in einem Neuronennetzwerk verantwortlich. Das Event-Delivery-System bietet noch weitere nützliche Funktionen, wie etwa das sofortige Ändern der Parameter. Außerdem ist es eine der wichtigsten Schnittstellen zu den inkludierten Integrations-/Feuerungsklassen. NEURONs Mechanismus im Umgang mit synaptischen Verbindungen basiert auf einem einfachen konzeptionellen Modell. Bei der Ankunft eines Impuls werden Neurotransmitter freigesetzt, welche im Gegenzug einige Mechanismen in der postsynaptischen Zelle beeinflussen (z. B. die Membranspannung oder einen weiteren Messenger). Dieser eben beschriebene Ablauf wird durch eine Differentialgleichung oder ein kinetisches Schema beschrieben. Das Einzige, was nun zählt, ist, ob ein Spike in der presynaptischen Zelle aufgetreten ist oder nicht. Dieses konzeptionelle Modell trennt die Spezifikation der Verbindungen zwischen den Zellen und der Spezifikation der Mechanismen, welche durch diese Verbindungen aktiviert werden.

Ein presynaptischer Impuls löst ein Event aus, welches nach einer kurzen Verzögerung die Fortleitung entlang des Axons und die Freisetzung von Transmittern bewirkt und schließlich beim postsynaptischen Mechanismus anlangt. Dort verursacht das vom presynaptischen Impuls ausgelöste Event eine Veränderung von Variablen, wie z. B. eine Änderung des Leitwerts.

Die Simulation mit Hilfe des Event-Delivery-Systems ist eine rechnerisch sehr effiziente Lösung. Das System berücksichtigt auch die diversen Verzögerungen zwischen der Entstehung und der Zustellung eines Events, die in verschiedenen Umgebungen unterschiedlich hoch ausfallen können. So ist die Anzahl der generierten Events pro Zeiteinheit selten gleich groß wie die Anzahl der empfangenen Events. Auch die Verzögerung kann beliebig lange sein. Auf Grund der Trennung der Spezifikation der Verbindungen zwischen Zellen und der postsynaptischen Mechanismen, die durch diese Verbindungen aktiviert werden, sind NEURON Modelle auch mit

anderen Event-Delivery-Systemen kompatibel. Ein Beispiel ist das „Parallel-Discrete-Event-Delivery-System“, das in NeoSim verwendet wird.

3.8.2. Integrationsmethoden

NEURON stellt mehrere verschiedene Integrationsmethoden [34] zur Verfügung. Es ist schwierig zu sagen, welche Integrationsmethode die jeweilige Simulation beste ist. Dies hängt von mehreren verschiedenen Faktoren ab, und jede Wahl einer Integrationsmethode beeinflusst entweder die Genauigkeit auf der einen Seite oder die Stabilität bzw. Laufzeit einer Simulation auf der anderen Seite. Oftmals ist es am besten, die geeignete Integrationsmethode an Hand empirischer Tests zu bestimmen.

NEURON bietet zwei Integrationsmethoden mit fixen Zeitschritten: Ein Rückwärtsdifferenzenverfahren (Backward-Euler) und eine Crank-Nicholson-Integrationsmethode. Das Crank-Nicholson-Verfahren ist eine finite Differenzenmethode zur numerischen Lösung von partiellen Differentialgleichungen. Es ist eine Integrationsmethode der zweiten Ordnung, und sie ist numerisch stabil, das heißt, dass sie gegenüber kleinen Störungen in der Datenmenge unempfindlich ist oder vielmehr, dass sich Rundungsfehler nicht zu stark auf die Berechnungen auswirken. Das Crank-Nicholson-Verfahren [35] ist der Durchschnitt des Forward-Euler-Verfahrens und des Backward-Euler-Verfahrens. Die Methode wurde von John Crank und Phyllis Nicholson in der Mitte des 20. Jahrhunderts entwickelt.

Der standardmäßige Integrator von NEURON ist der Backward-Euler-Integrator. Dieser ist ein Integrator erster Ordnung bezüglich der Zeit und produziert im Allgemeinen auch mit größeren Zeitschritten noch qualitativ ausreichende Ergebnisse. Wenn man das Verfahren jedoch mit sehr großen Δt einsetzt, wird der stabile Zustand eines linearen Modells bereits in einem Schritt berechnet und die Lösung sehr rasch an den stabilen Zustand eines nicht-linearen Modells angenähert. Um eine Genauigkeit der zweiten Ordnung zu ermöglichen, ohne nicht-lineare Gleichungen durchrechnen zu müssen, bietet das Crank-Nicholson-Verfahren (CN) keine fixen Zeitschritte sondern abgestufte Zeitschritte. Die Berechnungskosten eines einzelnen Zeitschrittes sind quasi dieselben wie beim Backward-Euler-Verfahren jedoch können viel schnellere Laufzeiten mit dem CN erreicht werden, weil das CN mit größe-

ren Δt für eine gegebene Genauigkeit funktioniert als das Backward-Euler-Verfahren. Der Nachteil der CN ist jedoch, dass es nicht bei Modellen eingesetzt werden kann, deren Status ausschließlich aus algebraischen Relationen bestehen. Weiters können störende Oszillationen auftreten, wenn das Δt zu groß ist, oder wenn im Modell schnelle Spannungsklemmen verwendet werden.

NEURON bietet auch einen Integrator, der mit variablen Ordnungen der Genauigkeit und mit variablen Zeitschritten umgehen kann. Dies ist der CVODE-Integrator (Cohen und Hindmarsh 1984). Modelle, die mit dem CN-Integrator gut funktionieren, laufen auch meistens mit dem CVODE-Integrator. Für eine bestimmte Laufzeit liefert der CVODE-Integrator üblicherweise eine höhere Genauigkeit. Bei einem Netzwerk mit künstlichen Neuronen ist der CVODE-Integrator meistens die beste Wahl.

3.9. Die Modell-Datenbank ModelDB

Ein großer Vorteil von NEURON ist die frei zugängliche Modell-Datenbank `ModelDB`. Hier gibt es eine Vielzahl fertiger Modelle, die man frei runterladen und in NEURON simulieren kann. Vor Allem für den Unterrichtseinsatz ist die Wichtigkeit dieser Datenbank hervorzuheben. Schüler und Studenten, die NEURON hauptsächlich zu Demonstrationszwecken nutzen wollen, haben hier einen Fundus an unzähligen Modellen, von einzelnen Neuronen bis zu ganzen Netzwerken, zur Verfügung. Aber auch für Forscher im wissenschaftlichen Bereich ist die Datenbank hilfreich. Sie ermöglicht einen Austausch und eine gemeinsame Arbeit an selben Projekten. Natürlich kann man hier auch seine eigenen Modelle publizieren.

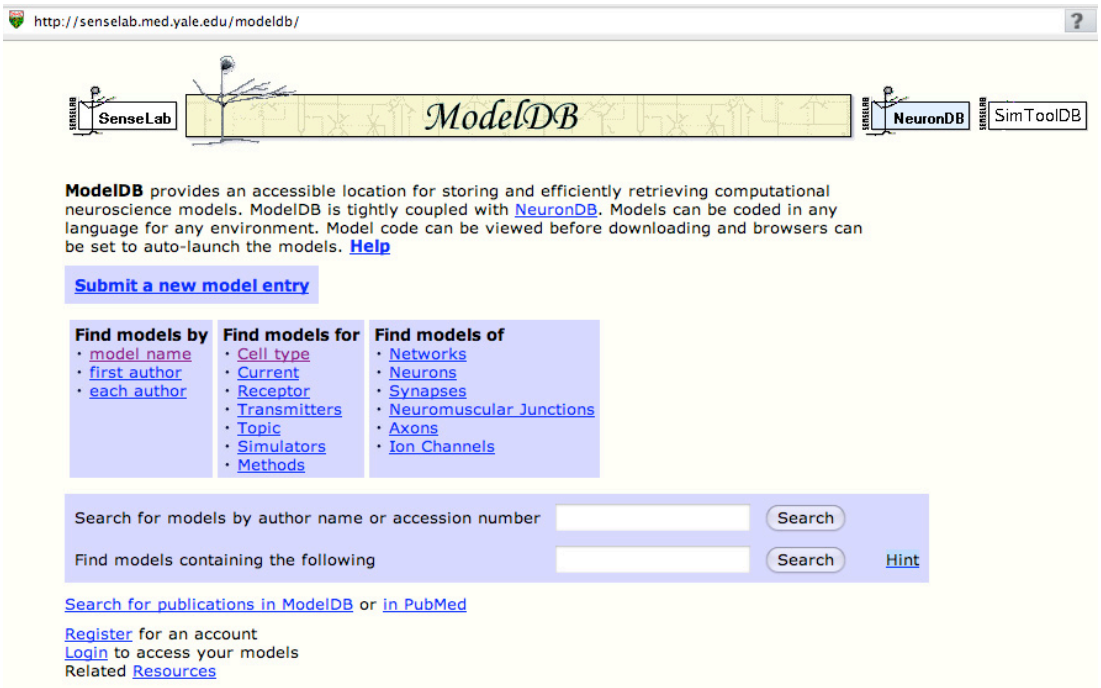

Bild 5: Screenshot der ModelDB

4. Weitere Simulationsumgebungen

4.1. GENESIS

GENESIS (GEneral NEural SImulation System) [36], [37] ist eine allgemeine Simulationssoftwareplattform, die speziell dafür entwickelt, wurde um die biologisch realistische Simulation von Nervensystemen zu ermöglichen. Das Einsatzgebiet reicht von der Simulation von subzellulären Elementen und biochemischen Reaktionen bis hin zur Simulation von komplexen Modellen einzelner Neuronen und großen Neuronennetzwerken. Auf Grund der objektorientierten Entwicklung von GENESIS und ihrer hochentwickelten Simulationssprache kann der Modellierer die Ressourcen des Simulators einfach erweitern und auch seine Modelle bzw. Modellkomponenten einfach wiederverwerten, bearbeiten oder austauschen.

4.1.1. Geschichtlicher Hintergrund

Ursprünglich wurde GENESIS von Dr. James M. Bower an der Caltech entwickelt. Seit Beginn war das Programm ausgelegt für die Simulation von großen realistischen Neuronennetzwerken, wie z. B.: das „Wilson and Bower piriform cortex“ Modell. Die erste Version erschien im Juni 1988. Die aktuelle Version 2.3 erschien am 17. März 2006 und ist zum freien Download auf http://www.genesis-sim.org/GENESIS/ verfügbar. Das Design von GENESIS wurde so gewählt, dass eine einfache Portierbarkeit auf Parallelcomputersysteme möglich ist. Das sogenannte „Parallel GENESIS“ (PGENESIS) ist eine Erweiterung von GENESIS und läuft auf beinahe allen parallelen Clustern, SMP Supercomputern, oder einem Netzwerk von Workstations die MPI oder PVM unterstützen.

4.1.2. Motivation der Entwicklung und Modellierungsphilosophie von GENESIS

GENESIS wurde als allgemeines erweiterbares Simulationssystem entwickelt, basierend auf den bekannten anatomischen und physiologischen Anordnungen von Neuronen und Nervennetzwerken (Bower, 1995; Bower und Beeman, 1998). Daraus folgt, dass Modelle einzelner Zellen in GENESIS üblicherweise eine dendritische Morphologie sowie eine Vielzahl von ionischen Leitwerten aufweisen. Diese Mo-

delltypen erfordern auf allen Analyseebenen eine detaillierte Gliederungsmöglichkeit. Die verbundenen Komponenten werden sodann zu größere Strukturen miteinander verbunden. Die daraus folgenden Verhaltensweisen werden schließlich numerisch vorausgesagt.

Die künftigen Fortschritte auf dem neurologischen Gebiet erfordern die Möglichkeit, Computermodelle zu entwickeln, die der aktuellen Anatomie und Physiologe der Nervensysteme selbst entsprechen. Basierend auf dieser Grundlage sollte das Projekt GENESIS folgende Funktionen unterstützen:

1. Modellbildung auf verschiedenen Ebenen von subzellularen Einheiten bis zu ganzen Systemen
2. Dem Modellierer die Möglichkeit zu geben, ihre Modelle weiterzuentwickeln und die Modelleigenschaften und –komponenten weiterzugeben
3. Geringe Abhängigkeit vom partikulären Computer (software-/hardwaremäßige Unabhängigkeit)
4. Ein verständliches graphisches Userinterface zur Verfügung zu stellen, um eine große Nutzerschicht zu erreichen
5. Die Nutzung in der Ausbildung mit dem Tool zu ermöglichen (u. a. in Universitätskursen)
6. Genügend Zeit und Ressourcen für den Benutzersupport zur Verfügung zu stellen

4.1.3. Überblick der Merkmale von GENESIS

GENESIS ist ein objektorientiertes Simulationssystem mit einer graphischen Benutzeroberfläche. Die Simulationen werden aus Modulen erstellt die einen Input haben, Berechnungen durchführen und daraus Outputs liefern. Neuronenmodelle werden aus diesen elementaren Komponenten wie Compartments (Einheiten) und variablen Leitfähigkeiten der Ionenkanäle erstellt. Zuerst werden die Compartments mit den Ionenkanälen und danach untereinander verbunden. So kann man Multicompartment-Modelle aller Komplexitäten erstellen. Auch die Neuronen können miteinander zu Netzwerken verbunden werden. Der objektorientierte Ansatz bürgt für die Flexibilität des Systems und ermöglicht den Modellierern einen einfachen Austausch und

eine Wiederverwertung der Modellkomponenten. Außerdem ist es möglich, die Funktionalität des Programms zu erweitern, ohne den Quellcode des Basisprogramms zu verändern.

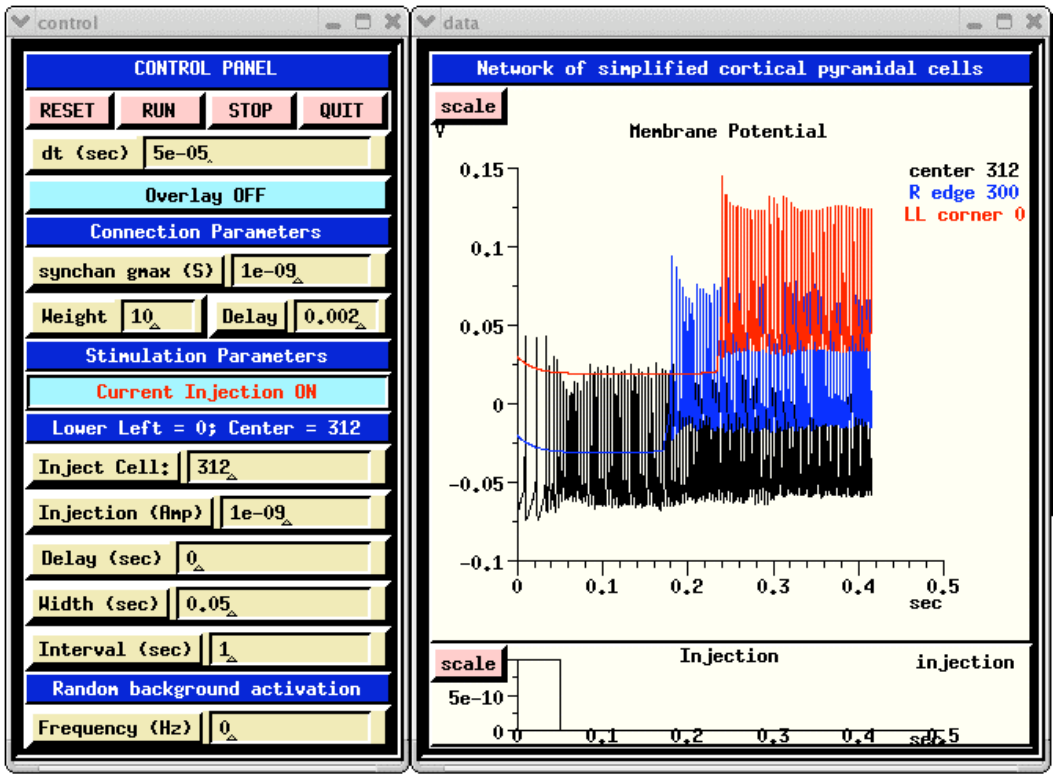

Bild 6: Beispiel einer GENESIS-Oberfläche

GENESIS verwendet eine hochentwickelte Simulationssprache, um Neuronen und deren Netzwerke zu bilden. Benutzerinteraktionen können über die Eingabezeile, über Skripts oder die graphische Oberfläche erfolgen. Eine Simulation setzt sich zusammen aus einer Sequenz von Kommandos der Skriptsprache. Das Tool und seine Module sind so dimensioniert, dass wenige Zeilen Code ausreichen, um anspruchsvolle Simulationen durchzuführen. Die Zusammensetzung und das Zusammenspiel der Komponenten von GENESIS, ist in **Bild 7** illustriert.

Die Ebene unter der Benutzeroberfläche nennt man den so genannten „Script Language Interpreter“ (SLI). Man kann diesen Interpreter mit einer UNIX-Shell vergleichen, die eine mannigfaltige Auswahl an Kommandos zur Erstellung, Kontrolle und Steuerung von Simulationen zur Verfügung stellt. Die graphischen Objekte und die Simulationsobjekte sind über die Skript-Sprache miteinander verbunden. Der Inter-

preter erlaubt interaktive Eingaben über die Tastatur (was ein interaktives Debugging und Steuern der Simulation ermöglicht) oder über eine Skript-Datei.

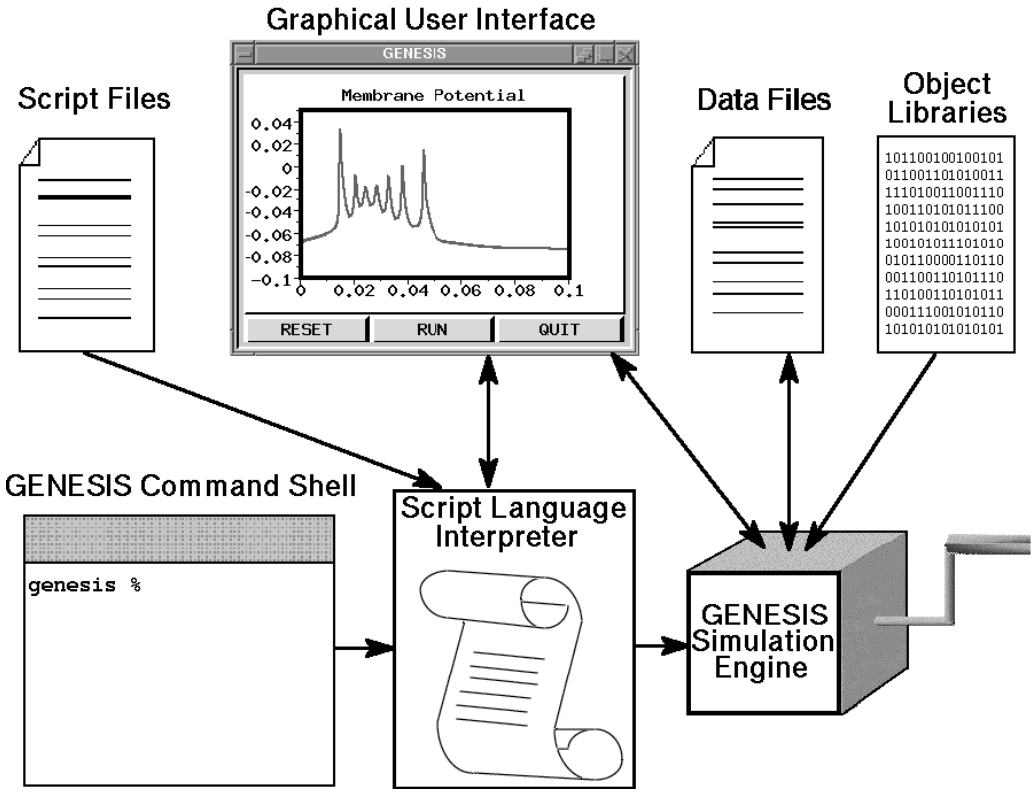

Bild 7: Zusammenspiel der Komponenten in GENESIS

Die „GENESIS Simulation Engine“ besteht aus dem Basiscode und stellt die nötigen Routinen für das Simulationssystem zur Verfügung. Dazu gehören die Ein- und Ausgabe sowie das numerische Lösen der Differentialgleichungen der Simulation von den verschiedensten neuronalen Objekten.

Neben der Eingabe über die Eingabezeile und die graphische Benutzeroberfläche ermöglicht die Simulations-Engine auch Simulationen mit Daten aus vorkompilierten GENESIS-Objekt-Bibliotheken. So ist beispielsweise die Konstruktion von komplizierten Neuronenmodellen einfach über eine Datendatei möglich, anstatt aufwändig über die Eingabezeile.

Die Objektbibliotheken von GENESIS beinhalten bereits Funktionen, um vielfältige Simulationen automatisch erstellen zu können. Dazu gehören kugelförmige und zy-

lindrische Compartments, mit denen die physikalische Struktur der Neuronen hergestellt werden kann. Weiters zählt man spannungs-/konzentrationsaktivierte Ionenkanäle, dendritische Kanäle und durch synaptische Inputs aktivierte Kanäle hinzu. Auch findet man verschiedene Arten von Synapsen. Es gibt zusätzlich auch Objekte zur Berechnung der intrazellulären Ionenkonzentration sowie der Ionendiffusion innerhalb einer Zelle. Auch andere biochemische Reaktionen können einfach modelliert werden. Natürlich gibt es auch verschiedenste Eingabeobjekte, wie Pulsgeneratoren und Spannungsklemmen sowie Messelemente.

4.1.4. Die graphische Benutzeroberfläche

Die GUI von GENESIS wurde mit XODUS (X-Windows Output and Display Utility for Simulations) entwickelt. Dies ermöglicht eine nutzerfreundliche Entwicklung und Beobachtung von Simulationen. XODUS bietet ein Set von graphischen Objekten, die aus der Nutzersicht den Berechnungsobjekten gleichen, außer dass sie eben graphische Funktionen ausführen. Wie auch bei den Berechnungsmodulen ist hier eine große Vielfalt an Darstellungsmöglichkeiten gegeben. Die Grafikmodule können über die Skriptsprache Funktionen aufrufen. Das bedeutet, dass die komplette Funktionalität der SLI auch über die Benutzeroberfläche zur Verfügung steht. So kann man in Echtzeit Parameter verändern und deren Auswirkung auch gleich beobachten. GENESIS bietet auch graphische Toolkits zur Erstellung und Visualisierung von Einzelzell-Modellen (Neurokit) und biochemischen Reaktionen (Kinetikit), wo noch kaum Skriptprogrammierung notwendig ist.
Bild 5 zeigt die anpassbare Oberfläche für Netzwerksimulationen.

4.1.5. Support und Dokumentation

Es existiert eine GENESIS Benutzergruppe, genannt BABEL [38]. Mitglieder dieser Gruppe sind dazu berechtigt, die E-Mail-Newsgroups zu abonnieren und auf die Diskussionsforen zuzugreifen. Hier findet man neue Simulationen, Bibliotheken von Zellen und Ionenkanälen, zusätzliche Simulatorkomponenten, Dokumentationen und Tutorials, Bugreports und Bufixes, wie auch eine allgemeine FAQ.

4.1.6. Ausblick und Entwicklungspläne

Mit der fortschreitenden Entwicklung der realistischen Modellierungsmethoden ist es ein Ziel von GENESIS, in der künftigen Version 3.0 eine umfangreiche Überarbeitung zur Verfügung zu stellen. Der Kern der Software wird mit einem moderneren Design neu implementiert in C++. Man erwartet sich dadurch Performanceverbesserungen, eine bessere Portabilität mit Windows und ein besseres Zusammenspiel mit anderen Modellierungsumgebungen. Das Entwicklungsteam von GENESIS nimmt auch am Projekt NeuroML (http://www.neuroml.org), gemeinsam mit Entwicklern von NEURON teil. Künftige Modelle sollen in einem gemeinsamen, simulatorunabhängigen XML-Format exportiert und importiert werden können.

4.1.7. Verfügbarkeit und Systemanforderungen

GENESIS ist lauffähig auf den gängigen UNIX-basierten Systemen wie Linux, aber auch Mac OS X. Unter Windows muss man die `Cygwin`-Umgebung verwenden.
Unter http://www.genesis-sim.org/GENESIS/ findet man den gesamten Quellcode inklusive Dokumentation als auch eine Vielzahl an Tutorials und Beispielsimulationen. Man findet hier auch schon vorkompilierte Versionen für Mac und Windows.
Das GENESIS Projekt wird durch das „U.S. National Institutes of Health“, „the Vice Chancelor for Health Affairs of the University of Texas System“, und „the U.S. NCRR“ unterstützt.

4.2. XPPAUT

Der Vollständigkeit halber sei hier auch das allgemeine numerische Tool XPPAUT [39], [40] (X-Windows PhasePlane plus Auto) erwähnt. Es wurde für die Lösung von Differentialgleichungen, Differenzengleichungen, verzögerten Gleichungen, funktionalen Gleichungen und stochastischen Gleichungen entwickelt.
Das Programm wurde ursprünglich von John Rinzel und Bard Ermentrout für die Illustration der Dynamik eines einfachen Modells mit einer erregbaren Membran entwickelt. Es entwickelte sich zu einem kommerziellen Produkt für MSDOS Computer und wurde PHASEPLANE genannt. Nun ist es als Programm für X11 und Windows erhältlich.

XPP enthält auch den Quellcode für das populäre Bifurkationsprogramm AUTO. Man kann zwischen XPP und AUTO wechseln und die Ergebnisse eines Programms im anderen weiter nutzen.

XPP unterstützt bis zu 590 Differentialgleichungen und ist portabel entwickelt worden, was die nur rudimentäre Oberfläche (entwickelt mit Xlib) begründet.

4.2.1. Features

XPP weist folgende Features auf:

- mehr als ein Dutzend verschiedene Solver
- bis zu 10 Grafikfenster können dargestellt werden
- Export als PostScript, GIF oder animated GIF möglich
- einfache Weiterverarbeitung gewährleistet (Histogramme, FFTs, ...)
- Berechnung von Equilibrien und linearen Stabilitäten möglich
- Poincaré-Abbildungen und Gleichungen an Zylindern
- Methoden für den Umgang mit gekoppelten Oszillatoren
- Gleichungen mit Dirac-Delta-Funktionen
- Animationstool zur Erstellung von einfachen Animationen aus den Simulationen
- Curve-Fitter basierend auf dem Marquardt-Levenberg-Algorithmus

4.2.2. Systemanforderungen

Die aktuelle Version von XPPAUT läuft unter der X-window Grafikumgebung und wird über die GNU pulic license vertrieben. Man findet dort vorkompilierte Binarys für die meisten Linuxdistributionen, Mac OS X und Windows.

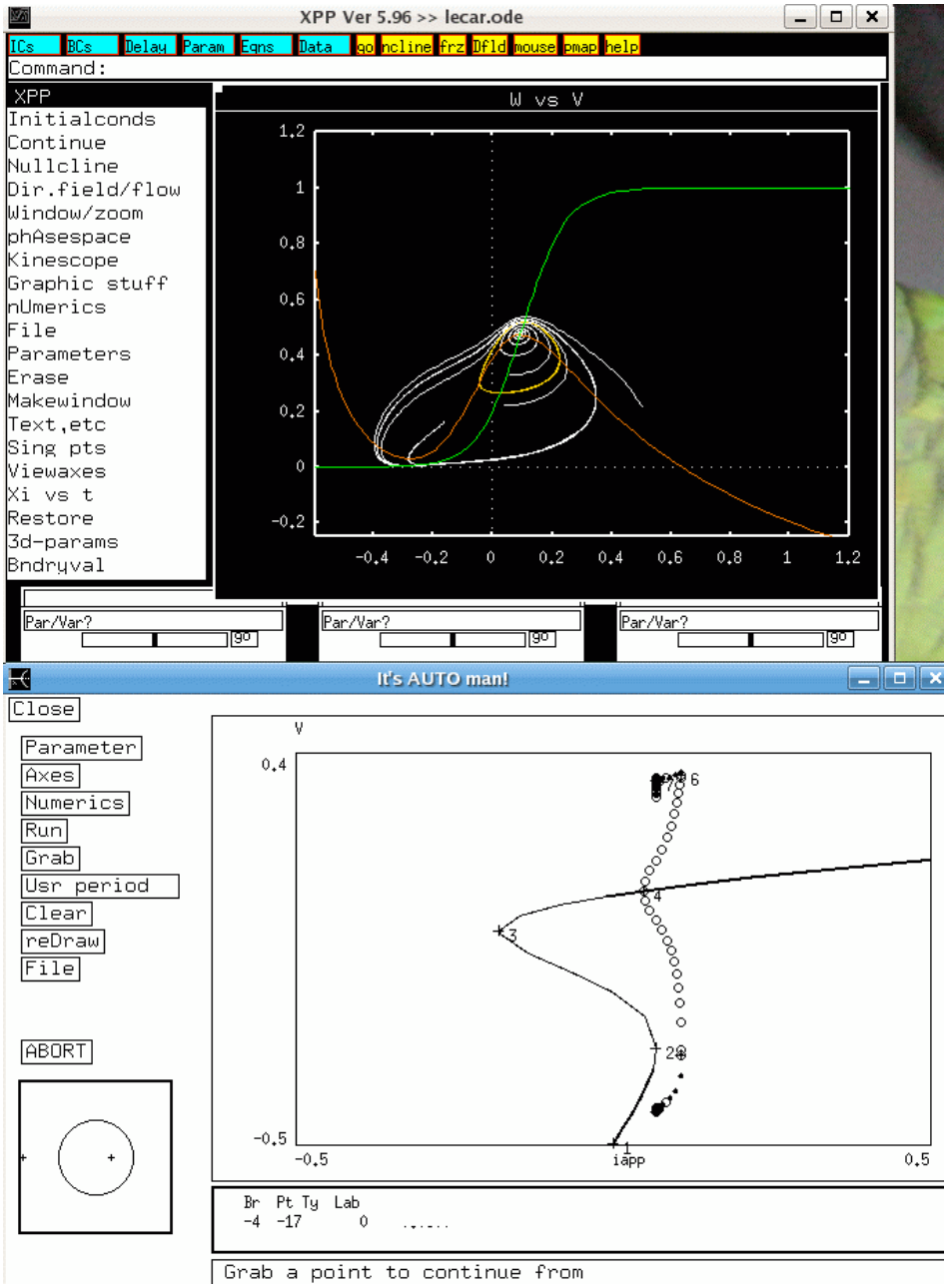

Bild 8: XPPAUT Interface mit AUTO Fenster

4.3. NEST (NEural Simulation Tool)

Das NEural Simulation Tool NEST [41], [42] wurde für die Simulation von großen heterogenen Netzwerken von Punktneuronen bzw. Neuronen mit wenigen Compartments entwickelt. NEST eignet sich im Besonderen für Modelle, die spezialisiert sind auf die Dynamik, Struktur und Größe des Nervensystems und eher weniger für detaillierte biophysikalische und morphologische Eigenschaften einzelner Neuronen.

Die Dynamik einzelner Neuronen wird als bekannt vorausgesetzt. NEST bietet die Möglichkeit, ganze Netzwerkarchitekturen einfach und flexibel zu erstellen, jedoch gibt es keine Möglichkeit, die internen Abläufe einzelner Neuronen zu verändern. Wenn man das tun möchte, muss man in den Kernel eingreifen (auf C++-Ebene) und den Kernel rekompilieren. Folgende Bereiche können vom Benutzer einfach erweitert werden:

- neue Modelle und Verknüpfungen können hinzugefügt werden (z. B. Neuronen und Synapsen)
- Bibliotheken mit nützlichen Programmen und Funktionen können hinzugefügt werden
- Erweiterbarkeit des Befehlssatzes der Simulatorsprache

Da NEST eine Kommandozeilenapplikation ist, muss man für die Simulation eine Skriptdatei erstellen, welche dann vom NEST Interpreter verarbeitet wird. Das Ergebnis ist meistens ein Datensatz, der dann mit Tools wie MATLAB oder Mathematica weiterverarbeitet und dargestellt werden kann.
NEST kann die Vorteile von Multiprozessor- und Multicore-Computern aber auch von Computer Clustern ausnutzen und skaliert auf diesen Systemen besser als im linearen Betrieb.

4.3.1. Das Modellierungskonzept von NEST

Knoten und Netzwerke

Ein Netzwerk besteht im Grunde genommen aus Knoten und deren Verbindungen. Knoten sind entweder Neuronen, Bausteine oder Unternetzwerke. Diese Knoten senden und empfangen Events verschiedener Arten, wie zum Beispiel Ströme oder Spi-

kes. NEST bietet eine große Auswahl an verschiedenen Neuronen, Bausteinen, Synapsen und Events. Darunter findet man die wichtigsten, wie:

- Neuronen-Modelle mit strom- und leitungsbasierten Synapsen
- Hodgkin-Huxley-Modelle
- statische Synapsen, Synapsen mit spike-abhängiger Verformbarkeit, Synapsen mit kurzfristiger Feuerung
- Bausteine zur Stromgenerierung (Gleich- und Wechselstrom), zufällige Aktionspotentiale, benutzerdefinierte Aktionspotentiale
- Bausteine zum Aufnehmen der Aktionspotentiale und Membranpotentiale

NEST bietet Funktionen, um große Netzwerke zu definieren und auch zu strukturieren. Mit Hilfe von Subnetzwerken hat man den Umgang mit großen Netzwerken erleichtert. Durch die modulare Bauweise kann der Benutzer auch einfach eigene Modelle erstellen.

Verknüpfungen

Eine Verknüpfung in NEST besteht aus einem sendenden Knoten, einem empfangenden Knoten, einer Gewichtung und einer Verzögerung. Die Gewichtung bestimmt die Intensität des Signals und die Verzögerung beschreibt die Übertragungsdauer von einem Knoten zum anderen. Pro Verknüpfung kann man jedoch nur einen Übertragungsmodus einstellen. Der Synapsenmodus zum Beispiel beschreibt, wie sich ankommende Events auf das post-synaptische Neuron auswirken.
NEST bietet diverse Funktionen zur Erstellung von angepassten Verknüpfungsschemata, wie zum Beispiel konvergente und divergente Verknüpfungen zwischen einzelnen Neuronen, zufällige Verknüpfungen und topologische Verknüpfungen zwischen den Neuronenschichten.

Events

Als Event wird in NEST eine Information definiert, die von einem Knoten (Neuron) zum anderen transportiert wird. Jeder Event besteht aus einem Datum der Erstellung (Zeitstempel) und einem Gewicht der Verknüpfung. Der Zeitstempel ist mit der Verzögerung der Verknüpfung kombiniert, um bestimmen zu können, wann der Event

am Zielknoten angekommen ist. Jedes Neuron besitzt Funktionen zur Annahme und Weiterverarbeitung von Events.
Je nach Art der übertragenen Informationen gibt es verschiedene Arten von Events. Die wichtigsten seien hier genannt:

- SpikeEvent: zur Übertragung von Aktionspotentitalszeiten
- CurrentEvent: um Gleich- oder Wechselspannungen zu übertragen
- PotentialEvent: zur Übertragung der Membranpotentialsdaten
- PreciseSpikeEvent: zur Übetragung von Aktionspotentialen außerhalb des Zeitrasters der Simulation

4.3.2. Verteilte Systeme

NEST kann die Vorteile von Multiprozessorsystemen und auch Computerclustern voll ausschöpfen. Auf Computerclustern verteilt NEST das Netzwerk auf die vorhandenen Rechner, sodass jeder Rechner seinen eigenen Teil des Gesamtnetzwerks besitzt. Auf Einzelplatzrechnern verteilt NEST das Netzwerk auf die zur Verfügung stehenden Prozessoren bzw. Prozessorkerne.

4.3.3. Genauigkeit von NEST

Ein gemeinsames Problem von zeitbasierten Simulatoren ist die Schrittweite und die Integrationsschrittweite bei dynamischen Gleichungen. NEST bietet folgende Mechanismen um diese Probleme zu umgehen:

- NEST hat keinen globalen Gleichungs-Solver sondern berechnet aufgrund des heterogenen Netzwerks die Zustandsgleichungen pro Knoten. Jeder Knoten kann somit den am besten passenden Algorithmus wählen und somit seinen Status aktualisieren.
- Für eine Vielzahl von Neuronenmodellen integriert NEST die dynamischen Gleichungen mit einer Methode genannt „Exact Integration“ von Rotter und Diesmann. Diese Methode integriert Systeme von linearen zeitunabhängigen Differentialgleichungen.
- In NEST kann die Schrittweite der Integration h unabhängig von der Simulationsschrittweite gewählt werden. Modelle, die Differentialgleichungen z. B.

mit dem Runge-Kutta-Verfahren approximieren, können das *h* eine Einheit kleiner einstellen als die Simulationsschrittweite Δ. Innerhalb von Δ kann man die Anzahl der Integrationen auch mit adaptiver Schrittweite einstellen.

4.3.4. Perfomance

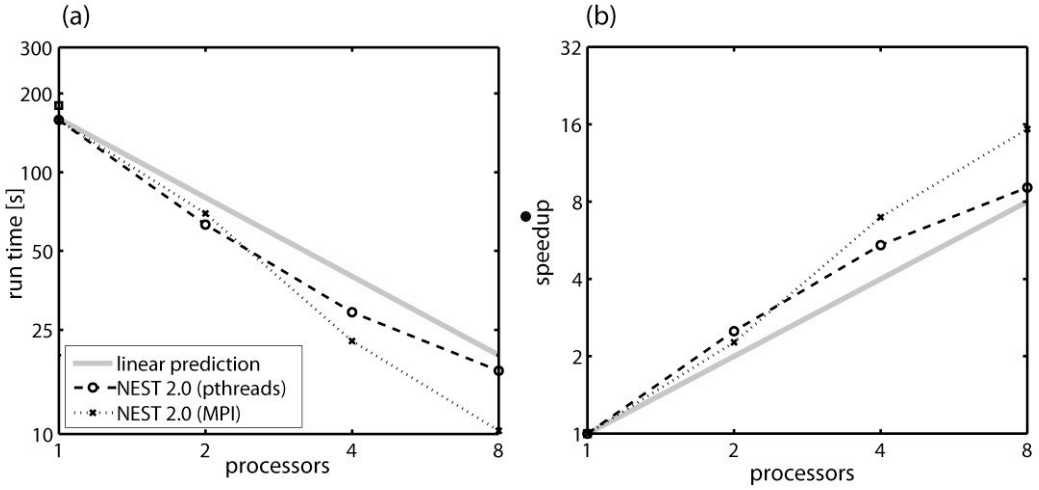

Bild 9: Performance-Vergleich in NEST

Die Skizze oben stellt ein Netzwerk aus 12500 Neuronen (80% stimulierend, 20% hemmend) dar. Jedes Neuron erhält Inputs aus 10% der gesamten Neuronen. Die Verknüpfungen injizieren Strom mit einer Verzögerung von *1 ms*. Die Anzahl der Synapsen betrug $15{,}6 \times 10^6$. Die Membranpotentiale der Neuronen wurden mit zufälligen Werten initialisiert. Das Bild zeigt, dass NEST besser als linear skaliert. Skizze a zeigt, wie die absolute Laufzeit mit der Erhöhung der Anzahl der Prozessoren fällt. Skizze b zeigt die Beschleunigung: mit vier Prozessoren ist die Simulation mehr als vier Mal schneller.

4.3.5. Entwicklung und Verfügbarkeit

NEST wird von der „Neural Simulation Technology Initiative (NEST Initiative)“, die im Jahre 2001 gegründet wurde, entwickelt. Momentan besteht sie aus folgenden Mitgliedern:

- Honda Research Institute Europe GmbH, Offenbach, Germany
- Brain Science Institute, RIKEN, Wako-shi, Japan
- Norwegian University of Life Sciences, Ås, Norway
- University of Freiburg, Germany

Es werden regelmäßig Distributionen zur Verfügung gestellt, die auch den kompletten C++ Sourcecode beinhalten. Unter http://www.nest-initiative.org kann man die Software gratis downloaden. Um das Tool ausführen zu können braucht, man einen aktuellen C++-Compiler und ein POSIX-kompatibles Betriebssystem. Es läuft problemlos auf Linux, Sun Solaris, Windows (unter `Cygwin`) und Mac OS X.

NEST ist ein interessantes Tool zur Simulation von großen Netzwerken und kann auf Grund seiner Spezialisierung in diesem Bereich gut punkten. Das Fehlen einer Oberfläche und die sehr technische Bedienung (z. B. Eingabe über Kommandozeile, Kompilierung notwendig, ...) erfordern aber umfangreiche Programmierkenntnisse. Dementsprechend liegt das Einsatzgebiet in anderen Bereichen als jene von NEURON und GENESIS.

4.4. MATLAB

Der Vollständigkeit halber sei hier auch das komplexe Mathematiktool MATLAB erwähnt. Es ist ein plattformunabhängiges und kommerzielles Softwaretool zur Lösung und Darstellung verschiedenster mathematischer Probleme. Die Software ist für Berechnungen von Matrizen ausgelegt. MATLAB wird mit einer proprietären Programmiersprache bedient, die aber sehr der Programmiersprache C ähnelt. MATLAB ist primär für die numerische Lösung von mathematischen Problemen entwickelt worden (im Gegensatz zur symbolischen Lösung anderer algebraischer Computersysteme). Aufgrund des leistungsstarken Kerns wird es in der Industrie und an Hochschulen hauptsächlich für die numerische Simulation sowie für die Datenerfassung, -analyse und -auswertung eingesetzt.

Diese Leistungsstärke prädestiniert MATLAB auch zur Berechnung von Simulationen aller Art. Mittels optionalen Toolboxen erhält man eine leistungsstarke und robuste Möglichkeit, Differentialgleichungen aller Art zu lösen. Somit ist es natürlich auch möglich, MATLAB für die Simulation von Nervenzellen einzusetzen. Im Endeffekt ist es ja gleichgültig, mit welchem Tool die Differentialgleichungen gelöst werden, insofern ist MATLAB für elementare Berechnungen sicher eine brauchbare Alternative. Wenn man das Tool ohnehin schon in Verwendung hat, kann man sich damit die Einarbeitung in ein anderes Softwarepaket ersparen.

Die Einsatzgebiete von MATLAB im Bereich der Nervensimulation sind aus diesen Gründen natürlich eingeschränkt bzw. nur unter erschwerten Umständen möglich. Die allgemeine Ausrichtung erschwert die Arbeit im Gegensatz zu spezialisierten Tools. Als Vorteil können aber die Mächtigkeit der Berechnungsengine, die mannigfaltigen Darstellungsmöglichkeiten, sowie das modellbasierende Design hervorgehoben werden. Jemand, der im Umgang mit der Simulationstoolbox schon vertraut ist, kann die Vorteile natürlich zur Gänze ausspielen.

Natürlich wird der Einsatz von der Art der persönlichen Verwendung abhängen, jedoch ist aufgrund der Bedeutung dieser Software in der allgemeinen Simulation eine Nennung obligatorisch.

4.4.1. Einsatz und Ausblick

MATLAB wird in den meisten technischen Studienrichtungen auf der gesamten Welt als Quasi-Standard eingesetzt. Insofern kann man grundlegende Kenntnisse damit schon als Voraussetzung ansehen. Durch die Plattformunabhängigkeit ist es auch auf den meisten Computern einsetzbar. Momentan wird an der Einführung eines „Distributed Computing Modells“ gearbeitet, um einen künftigen Einsatz auf den immer mehr verbreiteten Mehrkern- und Mehrprozessorsystemen zu ermöglichen.

MATLAB wird von „The MathWorks“ http://www.mathworks.com entwickelt und vertrieben.

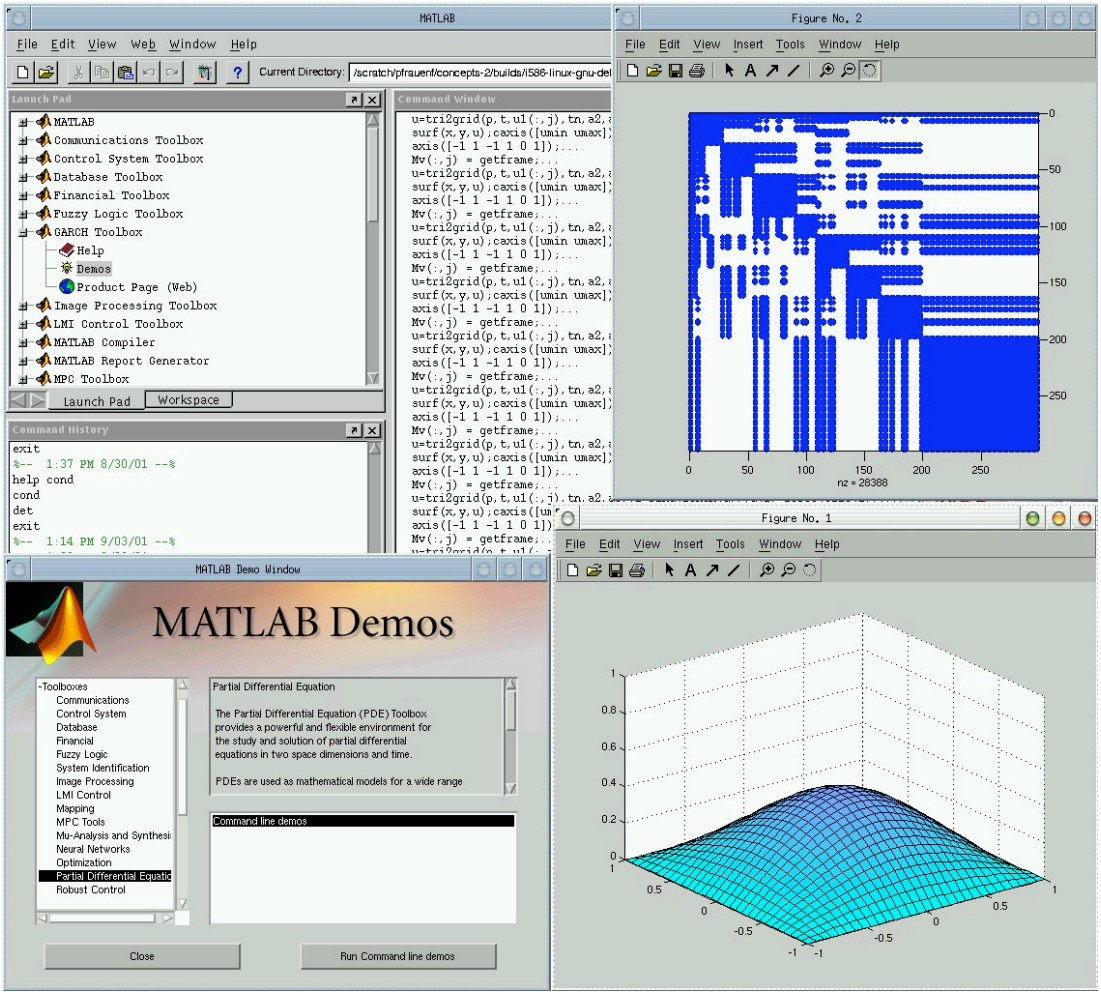

Bild 10: MATLAB Screenshot

5. NeuroML

5.1. Allgemeine Einführung

NeuroML [6], [44] ist eine gemeinschaftliche Entwicklung einer XML-basierten Modellierungssprache für die neurowissenschaftliche Modellierung. NeuroML stellt keine Standard-XML-Sprache dar sondern es ist eine Sammlung von verwandten XML-Projekten, die verschiedene Teile und Ebenen eines Nervensystems modellieren bzw. beschreiben können. Die Bandbreite reicht von der Beschreibung von intrazellulären Mechanismen und Ionenkanal-Kinetiken bis hin zur Modellierung von ganzen Neuronen und Neuronennetzwerken.

Mit der fortschreitenden Entwicklung werden auch die Modelle von Neuronen immer komplexer. Oftmals sind diese Modelle schon eigenständige Programme für sich und auch nur noch für den Forscher, der das Modell erstellt hat, verständlich. Um diese Modelle transparenter zu gestalten und somit auch zugänglicher zu machen, ist es sinnvoll, das Modell in strukturierter und deklarativer Form zu beschreiben. XML als verbreitete Beschreibungssprache eignet sich auch auf dem Gebiet der Neuronenmodellierung ideal für die Repräsentation und strukturierte Beschreibung der komplexen Modelle.

NeuroML verfolgt zwei große Ziele:

- Der Einsatz von deklarativen Spezifikationen von Neuronenmodellen mittels XML soll unterstützt werden.
- Die Entwicklung von weiteren XML-Standards im Bereich der Neurowissenschaften soll vorangetrieben werden.

5.2. Braucht man eine allgemeine Modellbeschreibungssprache?

Um die Simulation eines Modells, sei es eine Synapse, ein Neuron oder ein ganzes Neuronennetzwerk mit Hilfe eines Computers zu ermöglichen, ist es notwendig, das physikalische System zu konkretisieren und in ein Computerprogramm zu übertragen. Dieser Prozess bringt oft eine Verschleierung der Struktur und der maßgeblichen Details eines Modells mit sich, da man mit den jeweiligen Simulationssprachen oftmals Abstriche bei der Detailgenauigkeit machen muss.
Die Austauschbarkeit von Modellen wurde oftmals aufgrund folgender Faktoren erschwert:

a) Die schnelle Übertragung eines Konzepts in ein Modell vernachlässigt das Hervorheben der wichtigsten strukturellen Merkmale eines Modells. Man versteht oft nur schwer, welche Teile eines Simulationsskripts die Ausschlaggebenden für das Modell sind.
b) Für das Abbilden der Modelle können eine Vielzahl von Sprachen oder Simulationsprogrammen herangezogen werden. Diese Simulationsprogramme haben unterschiedliche Fähigkeiten und Kapazitäten, die untereinander jeweils nicht kompatibel bzw. austauschbar sind. Das heißt aber auch, dass Modellierer, die mit einer Sprache vertraut sind, nur selten auf eine andere Plattform umsteigen wollen.
c) Die verschiedenen Formalisierungen sind meist inkompatibel untereinander. Es ist auch selten der Fall, dass eine Simulationsumgebung mehrere Formalisierungsarten unterstützt.

Diese genannten Einflüsse führen dazu, dass ein Austausch von Modellen, eine gemeinsame Entwicklung und Diskussion nur schwer durchführbar ist.

5.3. Anforderungen an eine Modellbeschreibungssprache

In [44] werden die Anforderungen an eine Modellbeschreibungssprache erörtert. Sie sollen hier nochmals zusammengefasst werden.
Die Computersimulation von Nervenmodellen wird aus vielerlei Gründen betrieben. Einer der wichtigsten Gründe ist die Erforschung und das Testen von Hypothesen, die aus Konzepten von neuronalen Systemen abgeleitet wurden. Diese Entwicklung findet üblicherweise in Form von Beobachtungen, Experimenten und im Austausch mit anderen Forschern statt. Experimente versuchen die Vorhersagen (Hypothesen), die aus einem theoretischen Konzept abgeleitet wurden, zu verifizieren. In Bereichen, wo die Auswirkungen einer speziellen Konzeptualisierung nicht sofort augenscheinlich werden, bzw. wo man testen will, ob ein Konzept eine bestimmte Vorhersage abbildet, können Computersimulationen sehr hilfreich sein. Die Erstellung dieser Simulationen erfordert von den Forschern eine sehr rigorose und konkrete Beschreibung des Modellkonzepts, damit es auch von einem Computerprogramm berechnet werden kann. Dieser Schritt selbst darf nicht missachtet werden, da er oftmals Mängel im Modellkonzept aufzeigt.
Eine Modellbeschreibungssprache soll nach Goddard et al. [44] folgende drei Kerneigenschaften aufweisen:

- **Klarheit**
- **Portabilität**
- **Modularität**

5.3.1. Klarheit

Oftmals ist es schwierig, ein konkretes Modell in eine Skript- bzw. Programmiersprache zu übertragen. Vor allem die technischen Belange einer Programmiersprache können die eigentliche Struktur beeinflussen oder verfälschen. Um in diesem Punkt bestmögliche Effektivität zu erlangen ist es notwendig, eine einfache Beschreibungsform zu finden, die technische Schwierigkeiten beim Modellierungsprozess in den Hintergrund rückt und somit die Konzeptionalisierung einfacher und lesbarer macht.

5.3.2. Portabilität

Um eine Computersimulation überhaupt durchführen zu können, ist es nötig, die konkrete Form des Modells in einer Programmiersprache abzubilden bzw. in einer Form, welche in eine Programmiersprache übertragen werden kann. In einigen Fällen werden diese Konkretisierungen eines Modell in einer traditionellen Programmiersprache, wie zum Beispiel „C“, abgebildet. Um eine bessere Lesbarkeit bzw. Transparenz zu gewährleisten, wird in den meisten Fällen aber keine vollwertige Programmiersprache zur Modellabbildung verwendet, sondern eine andere Form, die in eine Programmiersprache übertragen werden kann. Üblicherweise handelt es sich hierbei um einfache Skriptsprachen. Solche Skriptsprachen – und somit Formen der Konkretisierung – gibt es zuhauf (eine pro Simulationsumgebung). Dies erschwert bzw. macht es beinahe unmöglich, Modelle gemeinsam zu nutzen oder in eine andere Simulationsumgebung zu übertragen. Innerhalb einer Wissenschaftsgruppe wird meist nur eine Simulationsumgebung genutzt. Der rege Austausch mit anderen Gruppen ist somit erheblich eingeschränkt. Aus diesem Grunde ist die Portabilität eine der Hauptanforderungen für die Nützlichkeit einer neuen, unabhängigen Modelbeschreibungssprache.

5.3.3. Modularität

Die Forschungen und Entwicklungen im Bereich der Neurowissenschaften befinden sich ja im regen Aufschwung und sind immer weiter am Wachsen. Deshalb kann angenommen werden, dass Computersimulationen eine immer größere Rolle spielen werden. Diese Simulationssysteme in den Neurowissenschaften sind hochkomplex, strukturiert, heterogen und oftmals nur schwer in ihrem kompletten Umfang zu durchschauen, was sich dann in der Abbildung der Modelle widerspiegelt. Die Bandbreite der Modellierungsmöglichkeiten und –notwendigkeiten im zeitlichen und räumlichen Bereich ist riesig. Die räumliche Dimension erstreckt sich über etwa sechs Größenordnungen vom Mikrometer- bis zum Meterbereich. Die zeitliche Achse erstreckt sich über etwa 15 Größenordnungen von Mikrosekunden bis zu Jahrzehnten. Einzelne Simulationssysteme decken oftmals nur einen gewissen Bereich dieser Bandbreite ab. Die Wissenschafter konzentrieren sich jedoch auf jeweils spe-

zifische Phänomene, die an verschiedenen Stellen der räumlichen und zeitlichen Achsen vorkommen und erforscht werden können. Um Beobachtungen von Phänomenen auch in anderen zeitlichen bzw. räumlichen Skalen ausführen zu können und schließlich untereinander zu vergleichen, besteht der Bedarf, die Modelle holistischer auszurichten – und somit für verschiedene Simulationsprogramme zugänglich zu machen. Dies fördert auch das Bedürfnis mit anderen Forschern zusammenzuarbeiten und Modelle auszutauschen.

Aus oben beschriebenen Gründen ergibt sich die unbedingte Notwendigkeit, dass eine neue Modellbeschreibungssprache/-methode auf jeden Fall eine hohe Modularität aufweisen soll, um eine bessere Zusammenarbeit zwischen verschiedenen Forschungsgruppen und Forschungsumfeldern zu gewährleisten, respektive überhaupt erst zu ermöglichen.

5.4. Modellbeschreibungen der Vergangenheit

In früheren Zeiten entstanden Computersimulationsumgebungen für neurowissenschaftliche Anwendungen meist basierend auf einem bestimmten Forschungsprojekt. In seltenen Fällen wurde darauf Acht genommen, dass man diese Simulationsumgebungen allgemein gestaltet und somit für künftige Erweiterungen öffnet sowie für eine größere Bandbreite an Simulationsanforderungen vorbereitet. Wenn es dann im Laufe der Zeit zu Erweiterungen kommen sollte, war das bestehende Simulationsprojekt aufgrund des nur rigiden und sehr spezifischen Designs nicht mehr vernünftig zu erweitern.

Schon frühzeitig sind allgemeine Programmiersprachen, wie z. B. „C“ oder „Fortran“, in den Hintergrund gerückt. Es haben sich auf dem Gebiet der neurowissenschaftlichen Simulationen skriptbasierte Modellbeschreibungssprachen durchgesetzt. Hierfür gibt es drei ausschlaggebende Gründe. Die meisten Programmiersprachen müssen zuerst kompiliert werden, wohingegen Skriptsprachen „nur“ interpretiert werden müssen. Das Kompilieren ist jedoch sehr zeitintensiv und gerade bei nur kleinen Änderungen des Modells sehr umständlich und langwierig. Weil das Erstel-

len des Modells meist gleich lange dauert wie die Simulation selbst, ist dies natürlich auch kostenintensiv.

Ein weiterer Grund, um auf allgemeine Programmiersprachen zu verzichten, ist die vermeintliche Unlesbarkeit des Quellcodes, vor allem für Laien auf dem Gebiet der Informationstechnik. Allgemeine Programmiersprachen haben einen hohen Anteil an Schlüsselwörtern und somit einen gewaltigen Überfluss an Text. Gleichzeitig fehlt die Unterstützung für spezifische neurowissenschaftliche Ausdrücke (z. B. Soma, Ionenkanal, usw.), was für den Neurowissenschafter abermals hinderlich ist. Wünschenswert ist es, eine Skriptsprache zu entwickeln, die mit der wissenschaftlichen Domäne eng verbunden ist. Die HOC-Sprache, die in NEURON zum Einsatz kommt sowie die SLI-Sprache die unter GENESIS verwendet wird, sind gute Beispiele von Skriptsprachen die auch für weniger geübte Informatiker lesbar und verständlich sind. Der dritte Punkt der Abneigung gegen allgemeine Programmiersprachen ist die Unklarheit, welche Sprachen auch weiterhin bestehen werden und für welche es auch in Zukunft Support geben wird.

Diese eben beschriebenen Gründe haben die Entwicklung von einigen neurowissenschaftlichen Skriptsprachen bewirkt. Wie schon beschrieben, haben diese Skriptsprachen ihre Wurzeln in kleinen, spezialisierten Projekten. Versuche, diese Skriptsprachen zu erweitern, um größere und diversifizierte Projekte zu simulieren, waren nur von mäßigem Erfolg gesegnet. Viele der Skriptsprachen unterstützen weder die notwendigen neuro-spezifischen Strukturen noch die Ausdrucksfähigkeit (z. B. Objektorientierung), die bei großen und diversifizierten vonnöten ist. Um bestehende, neue aber auch alte Modelle simulierbar zu machen, wurden bestehenden Skriptsprachen meist Erweiterungen hinzugefügt, anstatt die Sprachen neu zu entwickeln. Dass dies natürlich nur über unsaubere Umwege möglich ist, wird schnell deutlich.

5.5. Modellbeschreibungen der Zukunft

Die Hauptaufgabe der künftigen Modellbeschreibungssprache wird der Austausch von Informationen zwischen Softwarekomponenten eines Computersimulationssystems sein, mit der Hauptanforderung, dass die Softwarekomponenten, die für die Beschreibung, Simulation, Visualisierung und Beobachtung des Modells verantwortlich sind, direkt unterstützt werden.

Das Medium der Wahl zum Informationsaustausch zwischen den einzelnen Komponenten ist die Beschreibungssprache NeuroML. Diese Beschreibungssprache muss nicht zwangsläufig direkt manipuliert werden können oder überaus lesbar sein. Vielmehr sollte es entsprechende Userinterfaces oder andere passende Eingabemöglichkeiten geben, die wiederum ihrerseits den allgemeinen XML-Code generieren. Dieser muss die Komponenten, mit denen die Forscher hantieren, unterstützen und auch für künftige neue Konzepte erweiterbar sein.

Anhand der drei Grundanforderungen – **Klarheit, Portabilität, Modularität** – kann man jetzt die konkreten Anforderungen an diese allgemeine Modellierungssprache beschreiben.

5.5.1. Anforderung an die Klarheit

In einer Beschreibungssprache setzt sich der Begriff Klarheit aus drei wesentlichen Eigenschaften zusammen. Erstens sollte die Beschreibungssprache der Sprache des Neurowissenschafters möglichst ähnlich sein. Sie sollte der Sprache, die Neurowissenschafter zur mündlichen, textlichen oder graphischen Beschreibung von Neuronenmodellen verwenden, bestmögliche Unterstützung bieten. Zweitens sollte die Beschreibungssprache einfach und effektiv in ein Computerprogramm übersetzbar sein, welches auf den verschiedensten Plattformen lauffähig sein sollte. Drittens ist es unerlässlich, dass die Beschreibungssprache eindeutig ist. Das heißt, dass es nur eine Interpretation eines Ausdrucks in einem bestimmten Kontext geben kann.

Glücklicherweise schließen sich diese Eigenschaften nicht gegenseitig aus. Die erste Eigenschaft schlägt eine Sprache vor, die weitverbreitete Termini für bestimmte

Strukturen einsetzen soll wie, z. B. „Hodgkin-Huxley-Channel“ oder „Soma“. Die zweite Eigenschaft kann durch das Einbeziehen hochentwickelter State-of-the-art-Konzeptionalisierungsparadigmen erreicht werden. So kann man eine Beschreibung einer Neuronenpopulation etwa folgendermaßen konstruieren:

```
DefineNeuronPopulation (Name, CellType, Size, NumberOfCells)
```

Obgenannter Ausdruck kann in eine Vielzahl von verschiedenen Formen, passend für die diversesten Computerplattformen, übersetzt werden.
Verwendet man hingegen folgende Schreibweise:

```
for i = 1 to NumberOfCells
      CreateCell(CellType, Size)
```

wird es augenscheinlich, dass der Quellcode schon sehr spezifisch ist und nicht einfach auf andere Plattformen übertragen werden kann. Für parallele Computerplattformen ist er sogar schlichtweg ungeeignet. Hingegen kann das erste Konstrukt, `DefineNeuronPopulation`, bei passender Implementierung ohne weiteres auf Parallelrechnern ausgeführt werden. Die zweite Definition ist strikt sequentiell. Ohne weitere Analyse der `for`-Schleife ist es nicht möglich herauszufinden, ob die Neuronen unabhängig sind oder nicht.

Die dritte Eigenschaft kann durch die Verwendung einer präzisen Semantik herbeigeführt werden. Man muss eindeutige Definitionen haben, wie die verschiedenen Konstrukte sich gegenseitig beeinflussen und miteinander interagieren. Aus Sicht des Modellierers heißt das, dass man die Konstrukte eingehend verstehen muss und auch adäquat kommentieren soll.

5.5.2. Anforderung an die Portabilität

Die Anforderung an die Portabilität ergibt sich aus zweierlei Arten, nämlich dem Transfer des Modells von einer Computerplattform auf eine andere, als auch dem Transfer von einer Simulationsumgebung in eine andere. Ob ein Modell von einer

Computerplattform auf eine andere übertragen werden kann, hängt jedoch meist nur von der Verfügbarkeit der Simulationsplattform für die entsprechende Zielplattform ab.

Eine Überlegung, die hier untergebracht werden kann, und die mit der Implementierung in keinem direkten Zusammenhang steht, ist die Parallelisierbarkeit eines Modells. Ein Modell ist parallelisierbar, wenn es so beschrieben wurde, dass es von effizienten Algorithmen auf verschiedene Prozessoren verteilt werden kann und wenn es in einem Computer parallel konstruiert werden kann. Eine Möglichkeit, um ein Modell so parallelisierbar wie möglich zu beschreiben, ist die Verwendung einer Konzeptionalisierung auf höchster Ebene. Dann ist es möglich, die Implementierung an die jeweilige Computerplattform anzupassen.

Um ein Modell von einer Simulationsumgebung in eine andere zu transferieren, bedarf es schon größerer Anstrengungen. Portabilität zwischen zwei Simulationsumgebungen ist nur dann gewährleistet, wenn die Modelle zwischen den beiden Simulationsumgebungen bidirektional, eindeutig und ohne Verlust von Informationen transferiert werden können. Daraus folgend ergibt sich, dass es am einfachsten wäre, wenn verschiedene Simulationsprogramme dieselbe Sprache verwenden würden.

Das Hauptkriterium für die Portabilität ist also die Verfügbarkeit einer gemeinsamen Modellbeschreibungssprache, die von Beginn vom Design her für Erweiterungen aufgeschlossen ist. Nur so eine gemeinsame Sprache kann als Basis für einen Austausch von Modellen zwischen Forschergruppen oder mit Modelldatenbaken dienen.

5.5.3. Anforderungen an die Modularität

Die Neuronenmodelle werden immer komplexer und erstrecken sich über immer mehr strukturelle Ebenen. Dies erfordert eine hohe Modularität einer Modellbeschreibungssprache. Beispielsweise kann man das Wirken der Ionenkanäle mit Hilfe von Gleichungen beschreiben. Auf der anderen Seite haben bestimmte Gleichungen bereits einen Namen zugewiesen. Auch diese Form der Beschreibung sollte also unterstützt werden. Ein weiteres Beispiel ist die Beschreibung einer gesamten Neuro-

nennetzwerkpopulation. Diese kann man einerseits hierarchisch darstellen, andererseits auch wieder über Gleichungen beschreiben (z. B. die Interaktion zwischen einzelnen Zellen, die Abhängigkeit untereinander, usw.).

Ein modularer Aufbau einer Sprache beeinflusst auch die anderen beiden Kernanforderungen – Klarheit und Portabilität – positiv. Mithilfe der Modularität ist dann auch eine schrittweise Erweiterung der Sprache möglich: fehlende Beschreibungselemente können einfach hinzugefügt werden und sind somit in der gesamten Simulationsumgebung verwendbar.

5.6. Programmierungsfreie Modellbeschreibungen

Das Ziel programmierungsfreier Modellbeschreibungen, ist, eine Sprache zu erhalten, die sich vollkommen an den neurowissenschaftlichen Begriffen und Ausdrücken der theoretischen Modelle orientiert. Um dies zu erreichen wäre es denkbar, eine vorhandene objekt-orientierte Sprache wie etwa C++ oder Java heranzuziehen und diese mit spezifischen Klassen zu erweitern. Wenngleich man eigentlich eine Interpretersprache (z. B. Python) aufgrund der Klarheit bevorzugen würde, darf man nicht vergessen, dass bereits etablierte Sprachen eine Vielzahl von nötigen Softwarekomponenten bereits unterstützen. In unserem Fall sind dies eine integrierte Datenbankanbindung und die Unterstützung für graphische Benutzeroberflächen.

Eine künftige Sprache sollte fähig sein mit Datenbanken zu interagieren, um große Neuronennetzwerke effizienter verwalten zu können. Damit dies möglich ist, muss die neue Sprache vollkommen deklarativ sein. Es ist sehr unpraktisch, wenn man z. B. spezifische Suchanfragen mithilfe iterativer Programme ausführt – die sich ständig ändernden Abfragen würden zu keinem sinnvollen Einsatz führen.

Um nun eine Sprache vollkommen deklarativ zu erhalten, kann man benötigte iterative oder sequentielle Abläufe mit Namen versehen. Wenn man z. B. „Ranviersche Schnürringkette“ beschreiben will, wird im Hintergrund wohl eine Schleife den Aufbau selbiger bewerkstelligen. Für unsere ideale Modellierungssprache ist aber der

Name schon vollkommen ausreichend. Die Implementierung erfolgt dann in der jeweiligen Simulationsumgebung abhängig von den zugehörigen Gegebenheiten. Begriffe wie „`Soma`“, „`Dendrit`“, „`Synapsen`“, „`Leitfähigkeit`“, „`Ionenkanal`“ sind für so eine Beschreibungssprache natürlich ebenfalls unerlässlich.

5.7. Ist die Programmierung vollkommen ersetzbar?

Eine absolut deklarative Modellbeschreibung kann nur dann stattfinden, wenn jede mögliche Struktur eines Modells in der Beschreibungssprache benannt und somit codiert werden kann. Das Problem dabei ist, dass es in der Natur der Forschung liegt, neue Modelle bzw. Strukturen und Konstrukte zu suchen und zu erforschen. Eine Beschreibungssprache, die schon von Beginn an alle möglichen Strukturen verfügbar hat, wird wohl nur schwer die Sympathie der Forscher finden, weil dies ja technisch fast unmachbar ist. Es geht eher um die Frage, wie bereits etablierte Strukturen (z. B. „Hodgkin-Huxley-Kanal“) in der Modellbeschreibungssprache zur Verfügung gestellt werden sollen. Dies kann nur über Programmmodule geschehen, die jeweilige Strukturen über wohldefinierte Schnittstellen und Parameter implementieren.

Für neue Strukturen soll es die Möglichkeit geben, sie in einer Vielzahl von Programmsprachen abzubilden. Dies kann zu Beginn in einer Skriptsprache erfolgen – aus schon bekannten zeit- und Ressourcen sparenden Gründen. Wenn die geplante neue Modellstruktur ausgereift ist, kann sie auf eine kompilierbare Sprache umgestellt werden. Dies erfordert zwar einen gewissen Mehraufwand, bringt aber einige gravierende Vorteile mit sich. Wenn das Modell nicht mehr so oft verändert werden muss, wird es in kompilierter Form natürlich viel schneller ausgeführt als in Skriptform. Auch für die Distribution ergeben sich in kompilierter Form Vorteile. Man kann das Modell weitergeben, ohne aber die großen Geheimnisse der Struktur und dessen komplizierte Programmierung zu verraten. Auch die für die Verteilung notwendige ausführliche Dokumentation ist in kompilierbaren Programmen um ein Vielfaches einfacher und überschaubarer.

5.8. Alternative Modellbeschreibungen

Mit der fortschreitenden Entwicklung werden auch in wissenschaftlichen Simulationsumgebungen graphische Benutzeroberflächen immer häufiger. NEURON und GENESIS bieten zum Beispiel die Möglichkeit – von Neuronen bis zu Neuronennetzen – die Konstruktion über ein graphisches Benutzerinterface zu bewerkstelligen. Auch gibt es immer mehr das Bedürfnis, Modelle anhand von Daten aus Experimenten zu erstellen. Hierfür muss man seitens des Simulationstools Möglichkeiten zum automatisierten Einlesen von Daten bieten. Diesen beiden Tendenzen – weggerichtet von der textbasierten Eingabe auf Benutzerebene – muss natürlich Rechnung getragen werden. Der Benutzer verwendet also hauptsächlich die graphische Oberfläche, die wiederum den Quellcode des Modells generiert. In den Quellcode selbst wird häufig nur noch zur Feinjustierung eingegriffen bzw. dann, wenn die graphische Eingabemaske inadäquat für das aktuelle Modellierungsvorhaben ist. Bezüglich der Klarheit und Modularität sei hier erwähnt, dass es wünschenswert wäre, wenn Begriffe aus der graphischen Oberfläche so weit wie möglich auch in der textlichen Beschreibung abgebildet wären. Wenn zum Beispiel in der graphischen Oberfläche von einem „Dendrit“ gesprochen wird, sollte das auch in der textlichen Abbildung ein „Dendrit“ sein.

5.9. Der Aufbau von NeuroML

Der NeuroML Standard erstreckt sich über vier Ebenen. Er ist konzipiert worden, um einen Daten- und Modellaustausch, eine Datenbankerstellung, Modellpublikationen und ein einheitliches Datenformat zur Kommunikation zwischen verschiedenen Simulationsumgebungen zu ermöglichen.

5.9.1. Die Ebenen von NeuroML

Analog zum biologischen Aufbau eines Nervensystems wurde der NeuroML Standard auch in Ebenen entwickelt. Jede Ebene erweitert die vorhergehende um zusätzliche Funktionen.

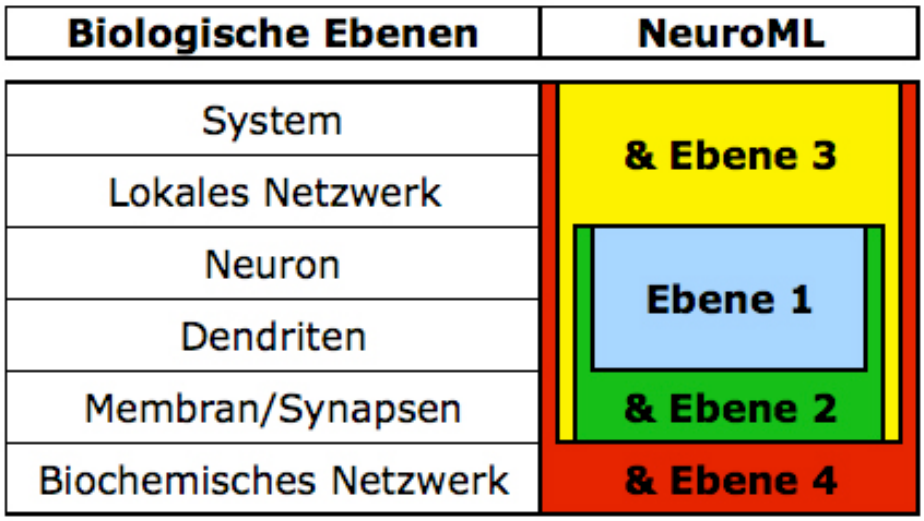

Bild 11: Ebenen in NeuroML

- **Ebene 1** stellt ein gemeinsames Format für die Beschreibung des neuronalen Aufbaus und der Metadaten bereit.
- **Ebene 2** baut auf Ebene 1 auf und stellt Beschreibungsmöglichkeiten für Membraneigenschaften, Dynamiken der Ionenkanäle, Synapsen und deren Verteilung parat.
- **Ebene 3** bildet zusätzlich die Verbindungen eines gesamten Neuronennetzwerkes ab.
- **Ebene 4** ist für die Beschreibung von subzellularen Prozessen gedacht (z. B. detaillierte Kalziumdynamiken) und ist noch in Entwicklung.

5.9.2. Verwendete Schemata in NeuroML

Für die verschiedenen Standards in NeuroML werden auch verschiedene Schemata verwendet. Folgendes **Bild 12** zeigt das Zusammenspiel der verwendeten Schemata in NeuroML:

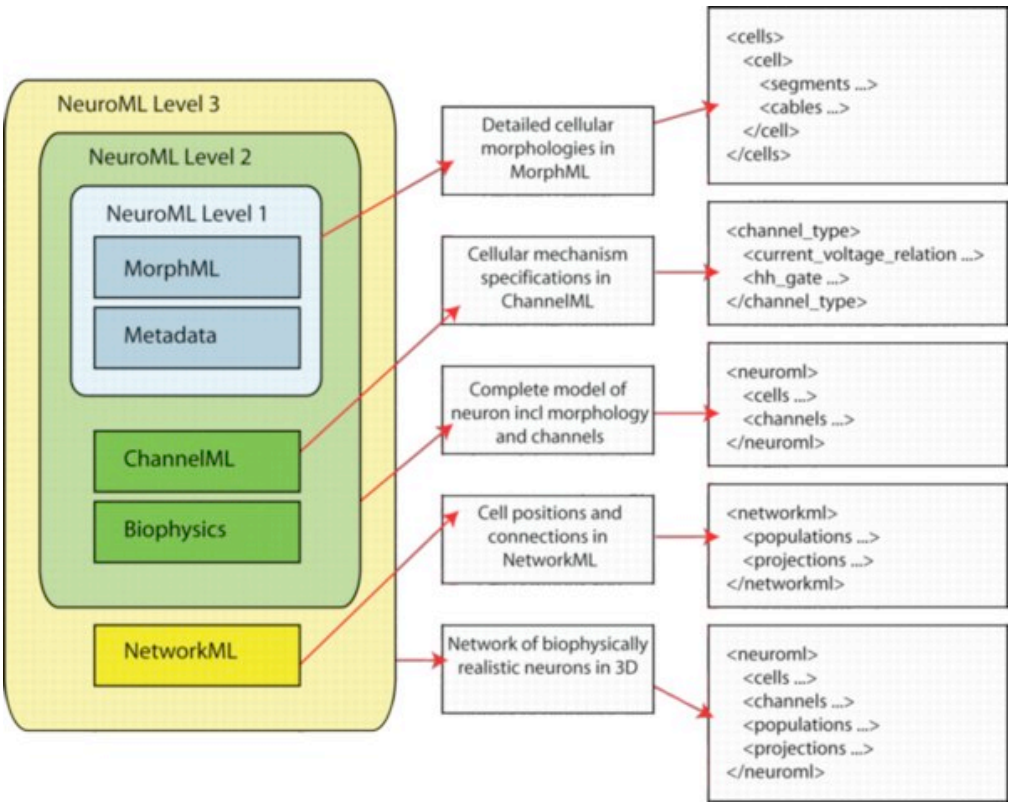

Bild 12: Zusammenspiel der Schemata in NeuroML

Die erste Ebene besteht aus einem Schema für Metadaten und einem `MorphML` Schema. Das Schema für Metadaten kann auf den aufbauenden Ebenen weiterverwendet werden. `MorphML` stellt die detaillierte Morphologie einer Zelle dar. Eine gültige/validierte `MorphML`-Datei soll künftig ins NEURON oder GENESIS Format übertragen werden können. Unter [45] findet man Tools, mit denen man Zellen, die der Ebene 1 und 2 entsprechen, bereits jetzt in NEURON importieren kann.

Ebene 2 stellt mit `Biophysics_v1.6` Mechanismen für die biophysikalischen Eigenschaften von Zellen bereit. Die Mechanismen von Synapsen und Ionenkanälen werden mit dem `ChannelML` Schema `ChannelML_v1.6` dargestellt. Validierte `ChannelML` Schema-Dateien sollen ebenfalls nach NEURON oder GENESIS übertragen werden

können. Die dritte Ebene befindet sich noch in einem frühen Entwicklungszustand. Sie soll die Spezifikationen der Zellpositionen und Netzwerkverbindungen abdecken. Das Kernschema der dritten Ebene ist `NetworkML_v1.6.`

5.10. Weitere Informationen

NeuroML befindet sich momentan noch in der Entwicklungsphase. So lange es noch nicht selbstverständlich von GENESIS oder NEURON unterstützt wird, wird ein großer Durchbruch bzw. ein allgemeiner Einsatz auf sich warten lassen. Aktuelle Neuerungen und Entwicklungen werden jedenfalls auf der Homepage http://www.neuroml.org bekannt gegeben. Dort findet man auch den Validator für XML-Dateien, sowie das NeuroML Development Kit.

6. Feldstudie – NEURON im Praxistest

6.1. Schüler und Studenten testen NEURON

In den vorhergehenden Kapiteln wurden einerseits die Grundlagen geschaffen, um die Funktionsweise von Nervenzellen und Neuronennetzwerken prinzipiell zu verstehen und andererseits um einen umfassenden Überblick in die Welt der Computersimulation zu erhalten. Das verbreitete Simulationstool NEURON wurde anhand eines Beispiels vorgestellt, und es wurden überdies die Interna eines Nervenzellensimulationspakets aufgezeigt. Weiters folgten Beschreibungen anderer Simulationsumgebungen sowie eine eingehende Diskussion von Anforderungen an eine allgemein gehaltene Beschreibungssprache im Bereich der Neuronenforschung. Angeknüpft an diese Diskussion folgte eine Auflistung des neuen XML-Standards NeuroML zur einheitlichen Beschreibung von Nervenzellen.

All dieses Wissen zusammenfassend und voraussetzend kam die Idee, das Tool NEURON von Schülern und Studenten testen und evaluieren zu lassen. Sie sollten zuerst auf eigene Faust und anschließend (wenn nötig) durch fachmännische Unterstützung die Simulationsumgebung kennen lernen. Die Schüler/Studenten sollten aus verschiedenen Wissensbereichen ausgewählt werden und auch verschiedene Vorkenntnisse aufweisen. Sie sollten versuchen, das Programm einerseits zum Laufen zu bringen und weiters die Simulationsumgebung anhand eines Beispiels zu testen. Hierfür sollte das Beispiel aus dem vorhergehenden Kapitel herangezogen werden. Weiters sollten sie versuchen, ein Modell aus der öffentlichen Datenbank ModelDB [5] zu laden und auch diese Simulation auszuführen.

Nach der Test- und Evaluierungsphase sollten die Tester ihre Eindrücke schildern. Interessant zu erfahren ist es, ob das Simulationsprogramm unterstützend in der Lehre und zum besseren Verständnis der Materie - also der Neuronenkunde - eingesetzt werden kann. In erster Linie soll hierbei die Bedienbarkeit und der Umgang mit dem Tool beschrieben werden. Ist es für Benutzer mit nur elementaren Computerkenntnissen überhaupt möglich, das Programm eigenständig zu bedienen? Lohnt sich der Aufwand, mit dem Simulationstool vertraut zu werden, im Gegensatz zum Nutzen,

den man als Student daraus ziehen kann? Sind die Oberfläche und das graphische Benutzerinterface wirklich verwendbar oder kommt man an einer Nutzung der Kommandozeile nicht herum? Sind Programmierkenntnisse für die adäquate Nutzung erforderlich? Resultierend aus diesen Betrachtungspunkten soll gemeinsam mit den testenden Schülern und Studenten eine Ist-Situation aufgezeigt werden und über sinnvolle Einsatzgebiete im Rahmen des Unterrichts nachgedacht werden. Daraus folgend sollen anschließend etwaige Verbesserungsmöglichkeiten formuliert werden.

6.2. Die Auswahl der Testpersonen

Wie bereits beschrieben, sollten die Testpersonen eine große Bandbreite abdecken und aus verschiedenen Bereichen mit unterschiedlichen Wissensständen und unterschiedlichen grundlegenden Computerkenntnissen ausgewählt werden. Insgesamt erklärten sich sieben Personen – davon 5 Schülerinnen und 5 StudentInnen – bereit, an dieser Evaluation des Nervensimulationsprogramms NEURON teilzunehmen.

6.2.1. Gruppe der Schülerinnen

Die beiden Schülerinnen besuchten zum Zeitpunkt der Evaluationsphase die siebte und die achte Klasse eines allgemeinen Oberstufengymnasiums. Aufgrund des Biologieunterrichts sind sie mit den anatomischen Hintergründen von Nervenzellen und deren Funktionsweise grundlegend vertraut. Der prinzipielle Aufbau und der Ablauf einer Informationsübertragung ist ihnen ein Begriff. Ihre Computerkenntnisse beschränken sich auf den Einsatz von Office-Programmen sowie Internet und E-Mail. Es sind keine Programmierkenntnisse oder sonstige tiefgründige Computerfähigkeiten vorhanden. Als Betriebssystem wird Windows XP verwendet.

6.2.2. Gruppe der Studentinnen

Die Gruppe der Studenten (bestehend aus drei Studenten und zwei Studentinnen; in Zukunft der Einfachheit halber als Studenten bezeichnet) besteht aus einem Biologen, einem Molekularbiologen, zwei Medizinern und einem Medizinischen Informa-

tiker. Alle Studenten sind mit der Anatomie und Funktionsweise von Nervenzellen vertraut – es ist ihnen möglich, die Ausgaben des Programms nachzuvollziehen. Die im ersten Kapitel beschriebenen biologischen Grundlagen können als Wissensstand vorausgesetzt werden. Bis auf einen Mediziner und den Informatiker haben die Studenten keine erwähnenswerten, über die alltäglichen Funktionen (Office, Internet, ...) hinausgehenden Informatikkenntnisse. Nur der Informatiker und ein Mediziner weisen Programmierkenntnisse auf. Die verwendeten Computerplattformen der Studenten sind Windows XP und Mac OS X.

6.3. Die Eindrücke der Schülerinnen

Da sich der Wissenstand der Schülerinnen sowohl im biologischen und anatomischen Bereich als auch im Computerbereich maßgeblich von dem der Studenten unterscheidet, ist es sinnvoll, die Eindrücke und Erkenntnisse hier gesondert zu evaluieren.

6.3.1. Umgang mit Neuron

Die Schülerinnen konnten die Simulationsumgebung NEURON problemlos aus dem Internet laden und auch auf deren Computer installieren. Nach dem Start des Simulationstools waren sie jedoch schon ein wenig überfordert. Sie konnten weder anhand der Beschreibung im vorhergehenden Kapitel noch mit den anderen Tutorials, die auf der NEURON-Homepage zu finden sind, eine lauffähige Simulation zustande bringen. Erst als man ihnen die beschriebene Simulation zum Laufen gebracht hatte und eingehend erklärte, konnten sie allmählich einzelne Parameter verändern und selbständig Simulationen berechnen lassen. Relativ schnell durchschauten sie die Veränderungen, wenn man zum Beispiel den Stimulusstrom erhöhte, die Länge der Dendriten veränderte oder die Größe des Somas anpasste. Nach einer Demonstration der einfachen Möglichkeiten des Tools fanden sich die Schülerinnen nach einer gewissen Zeit gut mit dem Programm zurecht. Nachdem man ihnen gezeigt hatte, wie man bereits fertige Modelle aus den öffentlichen Datenbanken runterladen kann und diese in NEURON simulieren kann, konnten sie auch dieses selbstständig bewerkstelligen.

Nach einer mehrstündigen und eingehenden Benutzung des NEURON-Tools (unter fachlicher Aufsicht und mit Hilfe) kamen die Schülerinnen zum folgenden Fazit:

Alleine hätten sie das Programm nicht bedienen können, und somit wäre es für sie nutzlos gewesen. Nach der fachlichen Einführung konnten Sie mit dem Programm umgehen und die komplizierten Prozesse eines Nervensystems besser verstehen. Das Programm war ihnen dabei behilflich, die Theorie, welche im Unterricht vermittelt wurde, zu illustrieren und verständlicher zu machen. Sie konnten sich den Aufbau einer Nervenzelle darstellen lassen und die einzelnen Elemente einer Zelle, wie den Zellkern, die Dendriten oder das Axon, schematisch anzeigen lassen. Das Anbringen einer Stromquelle und die daraus resultierende Feuerung eines Impulses konnten sie ebenso wahrnehmen. Vor allem die Ausbreitung des Impulses und die Weiterleitung über die Synapsen zu einer anderen Nervenzelle konnten sie auf diese Art und Weise anschaulich begutachten.

6.3.2. Negative Aspekte und Mankos

Als größtes Problem des Tools kritisierten die Schülerinnen die Benutzeroberfläche des Tools. Sie meinten, dass die Oberfläche nicht intuitiv bedienbar ist und auch nicht mehr zeitgemäß. Alleine hätten Sie sich damit nicht zurechtgefunden, und auch nach einer eingehenden Demonstration durch eine versierte Person hatten sie noch Schwierigkeiten im Umgang.

Insgesamt war für sie die Computersimulation im Allgemeinen auch ein wenig zu zahlenbehaftet. Zum besseren Verständnis der biologischen Abläufe im Nervensystem war es für die Schülerinnen verwirrend, mit all den verschiedenen Variablen jonglieren zu müssen. Da sie keine Programmierkenntnisse besitzen, war es für sie schwierig, den Überblick über die vielen Abkürzungen zu bewahren. Variablen wie `nseg`, `v` oder `gnabar_hh` sind für materienfremde Personen eher verwirrend als hilfreich. Auch die Werte dieser Variablen in Form von langen Gleitkommazahlen waren für die Schülerinnen fremd.

Da sie zwar die biologischen Grundlagen von Nervenzellen verstehen, nicht aber die elektrotechnischen Hintergründe und Abläufe – die ja für die Simulation von Model-

len ausschlaggebend sind – ist das gesamte Zahlenwerk für die Schülerinnen im Großen und Ganzen irrelevant. An der Oberfläche kritisierten sie die vielen eigenständigen Fenster, bei denen man schnell den Überblick verlieren konnte. Die Taskleiste war schnell überfüllt, und man konnte sich oft nur schwer zurechtfinden, wenn man z. B. von einem Diagramm zur schematischen Ansicht oder zur Steuerung der Simulation wechseln wollte.

Überdies bemängelten sie, dass es nicht wirklich möglich war zu erkennen, welche Werte bei den Eingaben noch sinnvoll sind. Ein Soma mit einem Durchmesser von 10 cm ist nämlich unrealistisch – ebenso eine Nervenzelle mit nur einem Dendrit und 50 Axonen; es wird vom Simulationstool dennoch eine Berechnung bzw. Simulation durchgeführt. Die Programmausgaben machen in diesen Fällen natürlich keinen Sinn mehr.

6.3.3. Verbesserungsmöglichkeiten von NEURON

Aus den vorher beschriebenen Kritikpunkten wurden gemeinsam mit den Schülerinnen folgende Verbesserungsmöglichkeiten vorgeschlagen:
In erster Linie wäre eine gänzlich neue und aufgeräumte Benutzer- und Eingabeoberfläche wünschenswert. Ziel wäre es, die vielen Fenster verschwinden zu lassen und ein geordnetes, zusammenhängendes und einheitliches Interface zu schaffen. Da sich auf diesem Gebiet des User Interface Design in letzter Zeit einiges getan hat, wäre eine Oberfläche in Anlehnung an die Office-Produkte bezüglich der Bedienbarkeit am sinnvollsten für die Schülerinnen. Ein weiterer Lösungsvorschlag wäre die Verwirklichung der Benutzeroberfläche mithilfe eines Internetbrowsers. So könnte die Plattformunabhängigkeit einfach gewahrt bleiben. Unterm Strich ist die jetzige Oberfläche einfach wenig einladend, um damit zu experimentieren und im weitesten Sinne spielerisch bzw. autark sich der Problematik der Nervenzellenstimulation bzw. Modellsimulation zu nähern. Dieser Aspekt der ziemlich veralteten Eingabeoberfläche von NEURON ist nicht nur für die Schülerinnen relevant sondern auch für alle anderen Nutzer. Es scheint so, als ob die Oberfläche nur zur Darstellung von Diagrammen sinnvoll eingesetzt werden kann. Jegliche Eingaben bzw. Modellierungs- und Erstellungsvorgänge sollten ohnehin über die Kommandozeile getätigt werden.

Natürlich ist es nicht so einfach für dieses komplexe Gebiet eine einfache Oberfläche aus dem Hut zu zaubern. Vor allem, da den Entwicklern des Tools wohl andere Prioritäten, wie etwa die Verbesserung der Simulationsengine, wichtiger erscheinen. Schließlich war ja das Programm nicht für den Einsatz im schulischen Bereich gedacht. Nichtsdestotrotz denke ich, dass es für die heutige Zeit angemessen wäre, eine passable, der Zeit entsprechende und einheitliche Oberfläche zu schaffen, da viele Bereiche davon profitieren würden.

Ein weiterer Wunsch wäre, die vielen Abkürzungen unter den Variablen zu bereinigen. So wäre es sehr praktisch, wenn man statt `v` auch „`Voltage`“ lesen könnte oder anstatt `nseg` etwa „`Number_of_Segments`“. In der internen Verarbeitung können und sollen ja die bisher vorhandenen Variablen bestehen bleiben. Für die Nutzung der Oberfläche wäre es aber praktisch, wenn man nicht andauernd nachschlagen müsste, welche Variablen welche Bedeutung haben. Dies könnte man auch einfach in eine neue Oberfläche integrieren. Das vorhandene NEURON könnte man relativ einfach erweitern, indem man z. B. eine Erklärung/Beschreibung der Variable eingeblendet bekommt, wenn man mit dem Mauszeiger über eine Variable fährt. Auch wäre es möglich, dass man die Simulationsumgebung anschließend in mehreren Sprachen zur Verfügung stellt. Diese Maßnahme könnte für die Schüler wahrscheinlich eine weitere Vereinfachung darstellen.

Die im vorherigen Absatz beschriebene Vielzahl von Variablen und deren lange Gleitkommazahlen wirkten auf die Schülerinnen verwirrend. Dieses Problem kann man nicht so einfach umgehen, da sich die Werte der Variablen oft nur in den hinteren Nachkommastellen verändern und somit für die wirklichen Simulationszwecke äußerst relevant sind. Eine denkbare Abhilfe wäre zum Beispiel, wenn man in Kombination mit der Einführung von sprechenden Variablenbezeichnungen auch eine Beschränkung, respektive Einstellung der Nachkommastellen, einführen könnte. So könnte für den schulischen Einsatz die Anzahl der Nachkommastellen auf die hinreichend notwendige Anzahl (etwa 2 – 3) beschränkt werden und für den wissenschaftlichen Einsatz die vollen Nachkommastellen verwendet werden. Diese Maßnahme würde die Übersichtlichkeit um einen zusätzlichen Faktor steigern und die Zahlenangst (Zitat: „So viele lange Zahlen!“) der Schülerinnen eindämmen. Eine weitere

Überlegung wäre, zusätzlich die Möglichkeit zu bekommen, Variablen ein- und auszublenden. In einer neuen Oberfläche könnte man so je nach Bedarf entweder mehrere Graphen generieren lassen oder mehrere Variablen anzeigen. So könnte man sich je nach Einsatzgebiet eine freundliche und übersichtliche Darstellung zurechtlegen. Im konkreten Fall des schulischen Einsatzes wird wohl die Anzeige von Variablenwerten eher in den Hintergrund rücken und sich das Gros der Anzeige auf Grafiken und Diagramme beschränken.

Das Problem der sinnlosen Eingaben in NEURON kann auch nicht so leicht abgewendet werden. Ein mögliches Gedankenkonstrukt wäre, eine Sicherheitsabfrage einzubauen, die bei bestimmten Werten nachfragt, ob man sicher ist, dass man mit diesen Werten simulieren will. Dies ist natürlich nicht so einfach, weil es ein sehr umfangreiches Feld abdecken muss und selbst eine gewisse künstliche Intelligenz voraussetzt. Für den schulischen Einsatz aber könnte man die Sicherheitsabfragen auf die wichtigsten Details beschränken. So könnte man Durchmesser, Längen und Anzahl von Soma, Dendriten und Axonen relativ einfach beschränken. Das Programm könnte etwa warnen, wenn man zu einem Soma 20 Axone hinzufügen will. Ähnlich könnte es mit dem Durchmesser sein, wenn ein bestimmter Wert über- oder unterschritten wird. Im wissenschaftlichen Einsatz wären solche Beschränkungen bzw. Warnhinweise wohl unnötig und auch kaum einführbar, da, wie schon erwähnt, eine Vielzahl von Variablen beeinflusst und Einstellungen getätigt werden. Im schulischen Demonstrationseinsatz, wo sich das meiste auf einige wenige Variablen beschränkt, wären solch elementare Sicherheitsabfragen denkbar und dem Verständnis der Schülerinnen sicher weiter hilfreich.

Ein Vorschlag, der von den Schülerinnen kam, war die Einführung von so genannten Templates bzw. Vorlagen. Ähnlich der Funktion „`Datei-Neu-Faxvorlage`“ in einem Textverarbeitungsprogramm wäre es wünschenswert und praktisch, wenn es in NEURON eine ähnliche Funktion gäbe. So wäre es praktisch, wenn man per „`Datei-Neu`“ bereits einfache Modelle per Knopfdruck generieren könnte. Man könnte die Anzahl der Dendriten und Axone auswählen, auf „`Erstellen`“ klicken und das einfache Modell könnte generiert werden. Vom Aufwand wäre diese Erweiterung sehr einfach zu bewerkstelligen. Im Prinzip bräuchte man nur den Code der

Tutorials in NEURON fix zu integrieren und auf Knopfdruck zu laden. Einige Einstellungen, wie etwa die Anzahl der Dendriten oder auch die Dimensionen der Komponenten, sollten schon im Vorfeld eingegeben werden können. Das Erstellen und Verbinden der Komponenten untereinander sollte sodann automatisch von statten gehen. Die detaillierte Anpassung kann dann ohnehin, wie sonst auch üblich, im Programm selbst vorgenommen werden. Wenngleich so eine Funktion im professionellen Einsatz größtenteils unnötig sein wird, wäre sie vor allem im schulischen und demonstrativen Einsatz hilfreich und von großem Vorteil. Es wäre gar nicht notwendig, eine Vielzahl von Modell-Templates anzubieten. Eine kleine Auswahl an elementaren Nervenzellen bzw. elementaren Nervenzellverbünden würde schon ausreichen. Bezüglich der Modelle könnte man sich einerseits an den schon vorhandenen Tutorials bedienen oder auch fertige Modelle aus der öffentlichen Datenbank heranziehen.

6.3.4. Resümee des Einsatzes von NEURON im Unterricht

Abschließend betrachtet kann man den Einsatz von NEURON im schulischen Bereich schon als vorteilhaft bezeichnen. Die Simulation von einfachen Nervenmodellen trug wesentlich zum besseren Verständnis der Materie (Funktionsweise von Nervenzellen) unter den Schülern bei. Wenngleich die Schülerinnen das Programm anfangs kaum selbst bedienen konnten, wäre es durch einen geübten und motivierten Lehrer keine große Schwierigkeit, einfache Modelle per Tutorial oder aus der vorhandenen Datenbank zu laden und diese den Schülern zu demonstrieren. Es hat sich auch gezeigt, dass nach einer guten und soliden Einführung die Schüler selbst mit dem Programm umgehen konnten und ein gutes Gespür für die Simulation erlangten. Wenngleich die oben beschrieben Mängel und vor allem die veraltete Oberfläche/Bedienung der Simulationsumgebung negative Aspekte darstellen, blieb bei den Schülerinnen unterm Strich ein positiver Eindruck bestehen. Der sonst trockene und nur wenig illustrative Unterrichtsstoff konnte anschaulich und animiert vermittelt werden. Es wurde ein neues Interesse für die Materie geweckt.

Die gemeinsam mit den Schülern identifizierten und beschriebenen Verbesserungen von NEURON werden wohl wünschenswert bleiben. Vor allem eine neue Oberflä-

che ist, wie beschrieben, nicht so einfach herstellbar und wohl auch nicht oberste Priorität der Entwickler. Nichtsdestotrotz könnte man einige der genannten Verbesserungsmöglichkeiten relativ einfach implementieren und hinzufügen, wie etwa die fixe Einbindung von einfachen Tutorials bzw. ein direkter Import aus den öffentlichen Modell-Datenbanken. Jede Verbesserung würde zur einfacheren und besser durchschaubaren Bedienung beitragen und somit den Einsatz im Unterreicht vereinfachen und praktikabler machen.

Vom Einsatz bzw. Test der anderen beschriebenen Simulationsprogrammen wurde abgesehen. Die meisten dieser Tools verlangen umfangreiche Programmierkenntnisse bzw. weisen gar keine Benutzeroberfläche auf und sind somit definitiv für den wissenschaftlichen Bereich vorgesehen. Ein Einsatz im schulischen Bereich – also zu Lehr- und Demonstrationszwecken – wäre mit diesen Simulationsumgebungen undenkbar.

6.4. Studenten testen NEURON

Wie bereits erwähnt, ist die Ausgangslage von Studenten eine ungleich andere im Vergleich zu jener der Schülerinnen. Deshalb wurden sie auch separat von den Schülerinnen mit der Evaluierung von NEURON beauftragt und auch getrennt davon analysiert. Bezüglich des biologischen Vorwissens über die Funktionsweise von Nervenzellen und Nervensystemen war die Studentengruppe relativ homogen. Alle ausgewählten Teilnehmer befanden sich zumindest im vierten Semester ihrer Studien. Somit konnte ein fundiertes Basiswissen vorausgesetzt werden. Auch der medizinische Informatiker wusste über die Materie bescheid, da er die LVA Nervenmodelle bereits absolviert hatte.

Bereits im Vorhinein kann erwähnt werden, dass die Studenten trotz einer etwas anderen Erwartungshaltung und besserer Vorkenntnisse zu einem sehr ähnlichen Ergebnis bei ihrer Bewertung von NEURON kamen. Aus diesem Grund werde ich hier nur noch jene Aspekte genauer beschreiben, die im Test mit den Schülerinnen nicht

aufgefallen bzw. genannt wurden. Die deckungsgleichen Nennungen werde ich nur kurz auflisten.

6.4.1. Eindrücke der Installation und der ersten Tests

Sowohl der Download des Tools von der Homepage als auch die Installation verlief bei allen Studenten reibungslos und ohne Probleme. Die eingesetzten Betriebssysteme waren Windows XP und Mac OS X „Leopard“. Auch bei der Ausführung des elementaren Beispiels hatten die Studenten weitgehend keine Probleme. Mit Hilfe des Tutorials konnten sie die erste Simulation nach kurzer Zeit zum Laufen bringen. Lediglich einer Studentin mit etwas weniger Computererfahrung musste eingangs ein wenig nachgeholfen werden. Es zeigte sich aber, dass Studenten mit Programmiererfahrung wesentlich schneller mit der Simulationsumgebung zurechtkamen als Studenten ohne Programmierkenntnisse. Dies liegt wohl darin begründet, dass man mit der Eingabe von Textbefehlen doch einfacher und besser zurechtkommt, als mit der Eingabe bzw. Modellierung über die Oberfläche. Die Studenten ohne Programmiererfahrung konnten den Quellcode, der in den Tutorials zur Verfügung gestellt wird, zwar kopieren und anschließend ausführen, jedoch war ihnen der Quellcode ohne Erklärung ein wenig undurchsichtig und fremd. Nach einer eingehenden Erklärung und Demonstration der Vorgehensweise war es nach kurzer Zeit kein Problem mehr, Parameter zu verändern und diverse Simulationen laufen zu lassen, Diagramme zu erstellen und verschiedene Werte abzufragen. Auch das Laden von Modellen aus der öffentlichen Datenbank stellte kein Problem dar. Alle Studenten meisterten diese Aufgabe ohne große Schwierigkeiten.

Nachdem einige elementare Simulationen berechnet wurden und auch verschiedene Modelle geladen und simuliert wurden, beschrieben die Studenten ihre Testphase und ihre Testeindrücke.

Wie bei den Schülern wurde die veraltete und unübersichtliche Oberfläche als größtes Manko der Simulationsumgebung dargestellt. Wenngleich NEURON unter den Nervensimulationsumgebungen wohl noch die beste Oberfläche aufweisen kann, ist diese aus Sicht der Studenten, die das Tool nur zu Lehr- und Anschauungszwecken nutzen wollen, nur rudimentär und nicht den aktuellen Standards entsprechend. Sie

meinten, dass die Oberfläche bei der Modellierung bzw. bei der Veränderung von Parametern eher eine Behinderung darstellt. Für die Anzeige von Daten und Diagrammen sei sie noch brauchbar; für die Erstellung, Modellierung und Veränderung von Parametern ist sie jedoch nicht intuitiv nutzbar und eher hinderlich bzw. nur umständlich und zeitverschwendend nutzbar. Die Eingabe über die Kommandozeile wurde nach einer Gewöhnungsphase als sehr praktisch, leistungsfähig und robust dargestellt. Diese Variante der Modellierung wurde nach kurzer Zeit die erste Wahl unter den Studenten.

Die Problematik der Variablenabkürzungen war bei den Studenten kein großes Thema. Relativ schnell hatten sie die gängigsten Variablen durchschaut und sich diese auch gut gemerkt, was sich im routinierten Umgang mit der Kommandozeile schnell zeigte. Nach einem Hinweis, dass die Schülerinnen eine eingebaute Erklärung der Variablen wünschten (etwa per erscheinendem Text am Mauszeiger), stuften sie diese Erweiterung zwar als hilfreich ein, aber nicht unbedingt notwendig für ihren Gebrauch und das bessere Verständnis der Materie.

Trotz einer guten Vorstellung der biologischen Abläufe und der Abläufe der Modellabbildung im Simulationsprogramm meinten die Studenten, dass sie eher weniger animiert wurden, selbst Modelle zu entwerfen und diese zu simulieren. Zum besseren Verständnis und zur Besserung der Vorstellungskraft der verschiedenen Abläufe in Nervenzellen und Nervensystemen war die Simulationsumgebung jedenfalls dienlich.

Schließlich bemängelten die Studenten, wie auch schon die Schüler, das Fehlen von eingebauten Templates für einfache Modelle. Zumindest die einführenden Tutorials hätten nach Ansicht der Studenten bereits in NEURON inkludiert sein sollen. Ebenso bemängelten sie die fehlende Unterstützung der öffentlichen Datenbanken. Hier wurde ein direkter Import bzw. Export von Modellen aus der öffentlichen Modelldatenbank gewünscht.

Ein Manko, das bei den Schülern unerwähnt blieb, war die Exportfunktion von Grafiken. Die Studenten waren verwundert, dass NEURON keine Funktion zum Export

bzw. zur Speicherung von Grafiken anbot. Die Möglichkeit, Diagramme nur per Bildschirmkopie speichern zu können, erschien den Studenten als sehr umständlich und als nicht mehr zeitgemäß. Diese Funktion wurde relativ häufig gebraucht und somit war dies ein großer Wunsch der Studenten, diese Funktion bereitgestellt zu bekommen.

6.4.2. Resümee des Einsatzes von NEURON bei Studenten

Wie bei den Schülern wurde hier an erster Stelle der Wunsch nach einer neueren Oberfläche geäußert. Wenngleich die Studenten auch relativ gut mit der Eingabe über die Kommandozeile zurechtkamen, war die schlechte Oberfläche dennoch ein Thema bei allen. Sie verstanden zwar, dass NEURON in erster Linie für den wissenschaftlichen Bereich entwickelt wurde und deshalb die Prioritäten eher im Bereich der Kommandozeileneingabe lagen, dennoch meinten sie, dass eine aufgeräumte und der Zeit entsprechende Oberfläche auch im wissenschaftlichen Bereich von Vorteil wäre.

Der Wunsch nach einer direkten Import-/Exportfunktion in die öffentlichen Datenbanken war bei den Studenten deshalb sehr groß, weil sie damit relativ häufig arbeiten und die Vielzahl der vorhandenen Modelle schätzen. Vor allem die Biologen waren an den verschiedenen Vorlagen sehr interessiert. Da sie eher weniger Informatikkenntnisse haben, hätten sie eine direkte Importmöglichkeit aus der Datenbank `ModelDB` sehr begrüßt.

Das Exportieren bzw. die Speichermöglichkeit von Diagrammen wurde von allen Studenten gleichermaßen gefordert. Diese Funktion sollte relativ einfach und ohne größeren Aufwand integrierbar sein. Dass es so eine Funktion noch nicht gibt, löste bei den Studenten Verwunderung aus. Schließlich ist man das Speichern von Bildern und Diagrammen aller Arten bereits gewohnt und erachtet es als selbstverständlich.
Vor allem die Biologen und Mediziner sahen den Einsatz von NEURON als hilfreich, zum besseren Verständnis von Nervenmodellen. Sie konnten sich auch vorstellen, das Programm weiterhin zu nutzen. Wie schon erwähnt, waren vor allem die Biologen sehr angetan von den vielen verschiedenen Nervenmodellen aus der öffentlichen Datenbank.

Um selbst Modelle zu generieren und dann auch zu veröffentlichen, sahen sich die Studenten jedoch nicht im Stande. Dies war aber auch nicht vorrangiges Ziel der Evaluierung von NEURON. Auch konnten die Studenten relativ gut mit den uneingeschränkten Möglichkeiten von NEURON umgehen. Sie hinterfragten sowohl die Ergebnisse der Simulationen kritisch als auch die Möglichkeit, beliebige Werte bei der Simulation eingeben zu können. Die Grenze der sinnvollen Eingaben für die diversen Parameter haben sie weitgehend im Auge behalten. Den internen mathematischen Abläufen schenkten sie aber wiederum großes Vertrauen. Da sie die Modelle der Simulation eher aus biologischer als aus mathematischer Sicht betrachteten, waren ihnen die Berechnungsabläufe und Integrationsmethoden von NEURON kein Begriff und auch nicht von großem Interesse. Hierbei vertrauten sie voll und ganz auf die Modellbauer und auf deren Wissen bezüglich der mathematischen Hintergründe.

Zusammenfassend kann man sagen, dass sich der Einsatz von NEURON bei Studenten sehr wohl als interessant herausgestellt hat. Wenngleich man sich an die Bedienung erst gewöhnen musste – das Simulationsprogramm unterscheidet sich doch maßgeblich von der Bedienung eines Internetbrowsers oder eines Office-Produkts – konnten die Studenten aus der Verwendung des Programms Vorteile ziehen und diese für sich nutzen. Die beschriebenen Mankos bzw. Verbesserungsvorschläge würden nach Ansicht der Studenten die Freude an der Nutzung und am Umgang mit der Simulationsumgebung NEURON jedenfalls massiv beeinflussen.

Nach den Erkenntnissen aus der Testphase von NEURON wurde entschieden, die übrigen Simulationsumgebungen nicht testen zu lassen. Sie hätten dem Fokus der Untersuchung nicht entsprochen, weil die Studenten das Tool vorrangig zu Lehrzwecken nutzen wollten. Alle anderen Umgebungen hätten ein umfangreicheres Vorwissen der Computersimulation verlangt. Da sich das Wissen der Studenten jedoch auf den biologischen Bereich der Nervenzellenstimulation beschränkte, ist ein Einsatz der übrigen Tools nicht sinnvoll. Diese sind eindeutig für den wissenschaftlichen Bereich gedacht. NEURON bietet die Vorteile einer großen Datenbank mit fertigen Modellen. Dies wurde bei den Studenten besonders geschätzt. Diese Anzahl an

frei zugänglichen Modellen sucht man bei den anderen Simulationsumgebungen vergeblich. Insofern ist ein Einsatz zu Demonstrations- und Lehrzwecken bei Studenten ohne Modellbildungserfahrung bzw. ohne Kenntnisse der Computersimulation als nicht sinnvoll zu erachten.

6.5. NeuroML

Die Idee, eine einheitliche Beschreibungssprache für Nervenzellen bzw. ganze Nervenmodelle zu schaffen, kann als sehr positiv erachtet werden. Wie im zugehörigen Kapitel beschrieben, würde eine strukturierte und plattformübergreifende Sprache eine Reihe von Vorteilen mit sich bringen. Zusammenfassend seien hier nur die beiden größten Vorteile, und zwar der Austausch der Modelle zwischen verschiedenen Plattformen (z. B. NEURON und GENESIS) und die bessere – auch von Menschen – Lesbarkeit genannt.

Die theoretischen Anforderungen an so eine allgemeine Sprache wurden ja eingehend erörtert. Ebenso wurde der aktuelle Stand der Entwicklung von NeuroML, das ja auf XML basiert, beschrieben. XML bietet den Vorteil, Informationen programmiersprachenunabhängig und strukturiert zu speichern. Aufgrund der Nähe zu HTML, seiner Einfachheit und auch guten Lesbarkeit, ist XML als Repräsentationssprache in der Informatik präsenter den je. Es hat sich zum Quasi-Standard der Datenrepräsentation bzw. Datenspeicherung entwickelt. Bei der Verwundung von XML ist man sozusagen gezwungen, seine Modelle deklarativ zu spezifizieren – im Gegensatz zu einem üblichen, proprietären Quellcode.

Die Speicherung der Daten im XML-Format bringt natürlich auch Nachteile mit sich. So ist aufgrund der hohen Redundanz des XML-Codes eine Bearbeitung dessen um einiges schwieriger und aufwändiger als die Bearbeitung eines normalen Quellcodes. Die Dateigrößen (wegen der hohen Redundanz) sind auch um einiges größer, als jene eines Binärcodes. Auch die Verarbeitung gestaltet sich aufgrund der Sperrigkeit des XML-Codes um einiges schwieriger als normale Textformate (z. B. `hoc`-Code in NEURON). Nichtsdestotrotz gibt es XML-Parsing-Tools für jede nennenswerte Pro-

grammiersprache. So muss man sich zumindest um die Bearbeitung des XML-Codes keine Gedanken machen. Üblicherweise ist XML auch nicht für die Aufnahme von Binärcode gedacht gewesen, weil dieser Code nicht mehr lesbar wäre, was aber einer XML-Spezifikation entspricht. Man kann dies aber mithilfe der BASE64-Codierung umgehen.

Will man aber eine einheitliche und strukturierte Sprache zum plattformübergreifenden Datenaustausch schaffen, kann man all diese genannten Nachteile vernachlässigen. Sie werden von den Vorteilen der Klarheit und strukturierten, deklarativen Form des Codes aufgehoben. Also ist die Idee der NeuroML, auf XML-Basis aufzubauen, jedenfalls zukunftsträchtig.

Heutzutage entsprechen Modelle von Nervenzellen im eigentlichen Sinne Computerprogrammen. Man kann sie leider nur lesen, wenn man die zugehörige Simulationsumgebung beherrscht. Insofern ist solchen Modellen ein Austausch mit anderen Forschungsgruppen verwehrt – kaum jemand wird sich die Mühe machen, ein weiteres Simulationstool zu erlernen. Die Vorteile der Lesbarkeit solcher Modelle würden sich auch im Lehreinsatz bewähren. Schüler oder Studenten, die nur wenig bzw. gar keine Programmiererfahrung haben, könnten von Nervenmodellen in Form von XML-Code (bzw. in Form der NeuroML) profitieren. Unabhängig davon, ob sie eine Simulatorsprache beherrschen, könnten sie nach einer gewissen Übung mit den Modellen umgehen. Welchen Simulator sie anschließend verwenden würden, wäre dann mehr oder weniger egal.

Zum Zeitpunkt dieser Arbeit wurde NeuroML von den gängigen Simulationstools noch nicht unterstützt. Erst die neuen Versionen von NEURON bzw. GENESIS werden die XML-Darstellung von Modellen beinhalten und NeuroML-Unterstützung anbieten. Auch die NeuroML selbst ist noch nicht zur Gänze implementiert. Ein Zeitpunkt der Fertigstellung dieser neuen Technologie bzw. der neuen Versionen der genannten Simulationsumgebungen kann zum Zeitpunkt der Erstellung dieser Arbeit noch nicht genannt werden.

7. Zusammenfassung – Ausblick

Abschließend betrachtet kann man ein positives Resümee aus dem Einsatz von NEURON zu Lehrzwecken in Schulen und unter Studenten ziehen. Es war eine interessante Erfahrung, die Schüler und Studenten bei der Evaluierung des Nervensimulationstools beobachten zu können, sie zu unterstützen und mit ihnen gemeinsam die Materie der Nervensysteme zu erkunden. Wenngleich die Bedienung von NEURON nicht die einfachste war, haben sich die Probanden nach kurzer Zeit und nach eingehender Erklärung gut mit dem Tool zurechtgefunden. Alle Befragten gaben an, dass sie Vorteile aus der Nutzung von NEURON ziehen konnten. Anschließend war ihnen die Materie besser klar, und sie konnten sich einen umfassenderen Überblick über die Welt und Funktionsweise der Nervenzellen verschaffen. Selbst Modelle zu erstellen und zu veröffentlichen war für die Testpersonen jedoch kein Thema. Das ist jedoch nicht weiter verwunderlich, da die Studenten dieses Thema weder in ihren Studien behandeln, noch besitzen sie die nötigen Kenntnisse der Computersimulation bzw. Modellierung. Eine einfache Nutzung der vorhandenen Modelle steht für die Gruppe der Testpersonen eindeutig im Vordergrund. Insofern wäre es wichtig, diesen Zugriff auf die Modelle in der öffentlichen Datenbank weiter zu vereinfachen.

Die gemeinsam mit den Schülern und Studenten im vorherigen Kapitel evaluierten und identifizierten Verbesserungen wären sehr wünschenswert. Sie werden jedoch mit ziemlicher Sicherheit ein Wunsch bleiben. Das Tool NEURON zielt trotz umfangreicher aber veralteter Oberfläche eindeutig auf die professionelle Nutzung, vorrangig im wissenschaftlichen Bereich, ab.

Man könnte im universitären Bereich den Einsatz solcher Simulationstools forcieren, um den Studenten einen besseren Einblick zu gewähren und eine Sensibilität für die Materie der Simulation zu schaffen.

Im Bereich der Forschung könnte NeuroML vieles vereinfachen: hier seien vor allem die Plattformunabhängigkeit, die bessere Lesbarkeit der Modellbeschreibungen und die Strukturiertheit genannt. Solange NeuroML aber noch nicht fertig gestellt ist und

auch noch nicht von den Simulationsumgebungen unterstützt wird, kann man den Einsatz und die wirklichen Vorteile jedoch nur schwer abschätzen.

Vor allem GENESIS hat sich mit dem Launch der dritten Version seines Modellierungstools einiges vorgenommen. Neben einer modernen, graphischen Oberfläche sollen eine XML-Repräsentation der Modelle, eine Rückwärtskompatibilität zur alten Version und auch Tutorials sowie Applikationen zu Lehrzwecken integriert sein. Wie viel davon wirklich implementiert werden wird und wann diese Version tatsächlich fertig gestellt sein wird, kann man zum jetzigen Zeitpunkt nicht abschätzen. Für das hier behandelte Thema des „Einsatzes von Nervensimulationstools zu Lehrzwecken“ wäre diese neue Version jedoch hochinteressant.

Man kann gespannt sein, in welche Richtung sich die Entwicklung in obgenanntem Bereich bewegen wird. Interesse an Tools, die einfach bedienbar sind und die Theorie anschaulich und beispielhaft vermitteln können, ist jedenfalls in großem Maße vorhanden.

Bilderverzeichnung

Literaturverzeichnis

[1] LVA's von Prof. Dr. Frank Rattay, Stand: Juni 2008
http://tuwis.tuwien.ac.at/zope/_ZopeId/60334587A3ZCapf6BxE/tpp/lv/search/processSearch?srchVal=rattay&aktion=Suchen

[2] LVA Nervenmodelle, Stand: Juni 2008
http://tuwis.tuwien.ac.at/zope/_ZopeId/60334587A3ZCapf6BxE/tpp/lv/lva_html?num=101186&sem=2008S

[3] NEURON an der Yale University, Stand: April 2008
http://www.neuron.yale.edu/neuron/

[4] NEURON an der Duke University, Stand: April 2008
http://neuron.duke.edu/

[5] ModelDB, Datenbank mit NEURON-Modellen, Stand: Juni 2008
http://senselab.med.yale.edu/modeldb/

[6] NeuroML, offizielle Homepage, Stand: Juni 2008
http://www.neuroml.org/

[7] Hodgkin-Huxley-Modell, Stand: Juni 2008
http://de.wikipedia.org/wiki/Hodgkin-Huxley-Modell

[8] XML, Eine Beschreibung des Standards, Stand: Juni 2008
http://de.wikipedia.org/wiki/Xml

[9] Die Nervenzelle, Stand: Juni 2008
http://de.wikipedia.org/wiki/Nervenzelle

[10] Dendriten, Stand: Juni 2008
http://de.wikipedia.org/wiki/Dendrit_(Biologie)

[11] Axon, Stand: Juni 2008
http://de.wikipedia.org/wiki/Axon

[12] Synapse, Stand: Juni 2008
http://de.wikipedia.org/wiki/Synapse

[13] Aktionspotential, Stand: Juni 2008
http://de.wikipedia.org/wiki/Aktionspotential

[14] TU BioMed, Gesellschaft für Biomedical Engineering an der TU-Wien, Prof. Dr. Frank Rattay
http://info.tuwien.ac.at/tubiomed/

[15] F. Rattay, M. Hofbauer, Mitschrift zur LVA Nervenmodelle, VO
http://publik.tuwien.ac.at/files/pub-tm_4743.pdf

[16] Nervensystem, Stand: Juni 2008
http://de.wikipedia.org/wiki/Nervensystem

[17] Gliazelle, Stand: Juni 2008
http://de.wikipedia.org/wiki/Gliazelle

[18] Neurotransmitter, Stand: Juni 2008
http://de.wikipedia.org/wiki/Neurotransmitter

[19] Membranpotential, Stand: Juni 2008
http://de.wikipedia.org/wiki/Membranpotential

[20] Nernst-Gleichung, Stand: Juni 2008
http://de.wikipedia.org/wiki/Nernst-Gleichung

[21] Goldman Gleichung, Stand: Juni 2008
http://de.wikipedia.org/wiki/Goldman-Gleichung

[22] Hodgkin-Huxley-Modell, Stand: Juni 2008
http://icwww.epfl.ch/~gerstner/SPNM/node14.html

[23] Spiking neuron models, Stand: Juni 2008
http://icwww.epfl.ch/~gerstner/SPNM/SPNM.html

[24] Hodgkin-Huxley-Modell, Stand: Juni 2008
http://de.wikipedia.org/wiki/Hodgkin-Huxley-Modell

[25] Hodgkin and Huxley Experiments, Stand: Juni 2008
http://physioweb.med.uvm.edu/cardiacep/EP/handh.htm

[26] Hodgkin and Huxley: Superheroes, Stand: Juni 2008
http://www.swarthmore.edu/NatSci/echeeve1/Ref/HH/HHmain.htm

[27] D. Schnitzer, Brainstorms – Eine Einführung in die Computational Neuroscience, Okt. 2003

[28] M. L. Hines, N. T. Carnevale, The NEURON simulation environment, Neural Computation 9, 1179-1209, 1997

[29] M. L. Hines, N. T. Carnevale, NEURON: A tool for scientists
Neural Computation 7, 123-135, 2001

[30] Documentation on NEURON, Stand: Juni 2008
http://www.neuron.yale.edu/neuron/docs/docs.html

[31] NEURON simulation environment in Scholarpedia, Stand: Juni 2008
http://www.scholarpedia.org/article/Neuron_simulation_environment

[32] N. T. Carnevale, M. L. Hines, The NEURON Book.
Cambridge, UK: Cambridge University Press, 2006

[33] N. T. Carnevale, M. L. Hines, Expanding NEURON's repertoire of mechanisms with NMODL, Neural Computation 12, S. 839-851, 2000

[34] Numerical ordinary differential equations, Stand: Juni 2008
http://en.wikipedia.org/wiki/Numerical_ordinary_differential_equations

[35] Crank-Nicholson Method, Stand: Juni 2008
http://en.wikipedia.org/wiki/Crank-Nicolson_method

[36] GENESIS (GEneral NEural SImulation System), Stand: Juni 2008
http://www.genesis-sim.org/GENESIS/

[37] GENESIS auf Scholarpedia, Stand: Juni 2008
http://www.scholarpedia.org/article/GENESIS

[38] BABEL - The GENESIS Users Group, Stand: Juni 2008
http://www.genesis-sim.org/GENESIS/whatisbabel.html

[39] XPPAUT auf Scholarpedia, Stand: Juni 2008
http://www.scholarpedia.org/article/XPPAUT

[40] XPPAUT Homepage, Stand: Juni 2008
http://www.math.pitt.edu/~bard/xpp/xpp.html

[41] NEST, neural simulation tool auf Scholarpedia, Stand: 2008
http://www.scholarpedia.org/article/NEST_(NEural_Simulation_Tool)

[42] NEST Homepage, Stand: Juni 2008
http://www.nest-initiative.org/

[43] MATLAB auf Scholarpedia, Stand: Juni 2008
http://www.scholarpedia.org/article/MATLAB

[44] N. H. Goddard, M. Hucka, F. Howell, H. Cornelis, K. Shankar, D. Beeman, Towards NeuroML: Model Description Methods for Collaborative Modelling in Neuroscience, Philosophical transactions of the Royal Society of London. Series B, Biological sciences, 1209 – 1228, 2001

[45] NeuroML Tools and Utilities, Stand: Juni 2008
http://www.morphml.org:8080/NeuroMLValidator/ToolsEtc.jsp